推酷

编程狂人

Programming Madman

NO.46

关于推酷

推酷是专注于IT圈的个性化阅读社区。我们利用智能算法, 从海量文章资讯中挖掘出高质量的内容,并通过分析用户 的阅读偏好,准实时推荐给你最感兴趣的内容。我们推荐 的内容包含科技、创业、设计、技术、营销等内容,满足 你日常的专业阅读需要。我们针对IT人还做了个活动频道, 它聚合了IT圈最新最全的线上线下活动,使IT人能更方便地 找到感兴趣的活动信息。

关于周刊

《编程狂人》是献给广大程序员们的技术周刊。我们利用技术挖掘出那些高质量的文章,并通过人工加以筛选出来。每期的周刊一般会在周二的某个时间点发布,敬请关注阅读。

本期为精简版 周刊完整版链接:http://www.tuicool.com/mags/5445126fd91b14511f001bf7

欢迎下载推酷客户端体验更多阅读乐趣

版权说明

本刊只用于行业间学习与交流署名文章及插图版权归原作者享有

目录

2014 非常好用的开源 Android 测试工具

译者：oschina

当前有很大的趋势是转向移动应用平台，Android 是最广泛使用的移动操作系统，2014 年大约占 80% 以上的市场。在开发 Android 应用的时候要进行测试，现在市场上有大量的测试工具。

本文主要是展示一系列的开源 Android 测试工具。每个工具都会有相应的简短介绍，还有一些相关的资源。Android 测试工具列表是按照字母来排序的，最后还会介绍几个不是特别活跃的 Android 测试相关的开源项目。

本文提到的开源 Android 软件测试工具包括：Android Test Kit, Android-JUnit4, Appium, calabash-android, Monkey, MonkeyTalk, NativeDriver, Robolectric, RoboSpock, Robotium, UIAutomator, Selendroid。

Android Test Kit

Android Test Kit 是一组 Google 开源测试工具，用于 Android 平台，包含 Espresso API 可用于编写简洁可靠的 Android UI 测试。

OSChina URL: http://www.oschina.net/p/android-test-kit

相关资源

* Android application testing with the Android test framework – Tutorial

(http://www.vogella.com/tutorials/AndroidTesting/article.html)

* Espresso for Android is here!

(http://googletesting.blogspot.ch/2013/10/espresso-for-android-is-here.html)

AndroidJUnit4

AndroidJUnit4 是一个让 JUnit 4 可以直接运行在 Android 设备上的开源命令行工具。

OSChina URL: http://www.oschina.net/p/androidjunit4

Appium

Appium 是一个开源、跨平台的自动化测试工具，用于测试原生和轻量移动应用，支持 iOS, Android 和 FirefoxOS 平台。Appium 驱动苹果的 UIAutomation 库和 Android 的 UiAutomator 框架，使用 Selenium 的 WebDriver JSON 协议。Appinm 的 iOS 支持是基于 Dan Cuellar's 的 iOS Auto. Appium 同时绑定了 Selendroid 用于老的 Android 平台测试。

OSChina URL: http://www.oschina.net/p/appium

相关资源
* Appium Tutorial
(https://docs.saucelabs.com/tutorials/appium/)
* Android UI testing with Appium
(https://blog.codecentric.de/en/2014/05/android-ui-testing-appium/)

Calabash-android

calabash-android 是一个基于 Cucumber 的 Android 的功能自动化测试框架。Calabash 允许你写和执行，是开源的自动化移动应用测试工具，支持 Android 和 iOS 原生应用。Calabash 的库允许原生和混合应用的交互测试，交互包括大量的终端用户活动。Calabash 可以媲美 Selenium WebDriver。但是， 需要注意的是 web 应用和桌面环境的交互跟触摸屏应用的交互是不同的。Calabash 专为触摸屏设备的原生应用提供 APIs。

OSChina URL: http://www.oschina.net/p/calabash-android

相关资源

* A better way to test Android applications using Calabash

(http://blog.teddyhyde.com/2013/11/04/a-better-way-to-test-android-applications-using-calabash/)

* Calabash Android: query language basics

(http://krazyrobot.com/2014/04/calabash-using-query/)

Monkey

Monkey 是 Google 开发的 UI/应用测试工具，也是命令行工具，主要针对压力测试。你可以在任意的模拟器示例或者设备上运行。Monkey 发送一个用户事件的 pseudo-random 流给系统，作为你开发应用的压力测试。

OSChina URL: http://developer.android.com/tools/help/monkey.html

MonkeyTalk

MonkeyTalk 是世界上最强大的移动应用测试工具。MonkeyTalk 自动为 iOS 和 Android 应用进行真实的，功能性交互测试。MonkeyTalk 提供简单的 "smoke tests"，复杂数据驱动的测试套件。MonkeyTalk 支持原生，移动和混合应用，真实设备或者模拟器。MonkeyTalk 使得场景捕获非常容易，可以记录高级别，可读的测试脚本。同样的命令可以用在 iOS 和 Android 应用上。你可以记录一个平台的一个测试，并且可以在另外一个平台回放。MonkeyTalk 支持移动触摸和基于手势交互为主的移动体验。点击，拖拽，移动，甚至是手指绘制也可以被记录和回放。

OSChina URL: http://www.oschina.net/p/monkeytalk

相关资源

* Using MonkeyTalk in AndroidStudio

(http://prativas.wordpress.com/2014/08/15/using-monkeytalk-in-android studio/)

NativeDriver

NativeDriver 是 WebDriver API 的实现，是原生应用 UI 驱动，而不是 web 应用。

OSChina URL: http://www.oschina.net/p/nativedriver

Robolectric

Robolectric 是一款Android单元测试框架，使用 Android SDK jar，所以你可以使用测试驱动开发 Android 应用。测试只需几秒就可以在工作站的 JVM 运行。Robolectric 处理视图缩放，资源加载和大量 Android 设备原生的 C 代码实现。Robolectric 允许你做大部分真实设备上可以做的事情，可以在工作站中运行，也可以在常规的 JVM 持续集成环境运行，不需要通过模拟器。

OSChina URL: http://www.oschina.net/p/robolectric

相关资源

* Better Android Testing with Robolectric 2.0

(http://corner.squareup.com/2013/05/robolectric-two-point-oh.html)

* Using Robolectric for Android testing – Tutorial

(http://www.vogella.com/tutorials/Robolectric/article.html)

RoboSpock

RoboSpock 是一个开源的 Android 测试框架。提供简单的编写 BDD 行为驱动开发规范的方法，使用 Groovy 语音，支持 Google Guice 库。Robo-Spock 合并了 Robolectric 和 Spock 的功能。

OSChina URL: http://www.oschina.net/p/robospock

相关资源

* RoboSpock – Behavior Driven Development (BDD) for Android

(http://www.methodsandtools.com/tools/robospock.php)

Robotium

Robotium 是一款国外的Android自动化测试框架，主要针对Android平台的应用进行黑盒自动化测试，它提供了模拟各种手势操作（点击、长 按、滑动等）、查找和断言机制的API，能够对各种控件进行操作。Robotium结合Android官方提供的测试框架达到对应用程序进行自动化的测 试。另外，Robotium 4.0版本已经支持对WebView的操作。Robotium 对Activity，Dialog，Toast，Menu 都是支持的。

OSChina URL: http://www.oschina.net/p/robotium

相关资源

* Robotium – Testing Android User Interface

(http://www.methodsandtools.com/tools/robotium.php)

* Android user interface testing with Robotium – Tutorial

(http://www.vogella.com/tutorials/Robotium/article.html)

UIAutomator

uiautomator 测试框架提高用户界面（UI）的测试效率，通过自动创建功能 UI 测试示例，可以在一个或者多个设备上运行你的应用。

OSChina URL: http://www.oschina.net/p/uiautomator

相关资源

* Automatic Android Testing with UiAutomator

(https://software.intel.com/en-us/android/articles/automatic-android-testing-with-uiautomator)

Selendroid

Selendroid 是一个 Android 原生应用的 UI 自动化测试框架。测试使用 Selenium 2 客户端 API 编写。Selendroid 可以在模拟器和实际设备上使用，也可以集成网格节点作为缩放和并行测试。

OSChina URL: http://www.oschina.net/p/selendroid

相关资源
* Mobile Test Automation with Selendroid

(http://www.methodsandtools.com/tools/selendroid.php)
* Road to setup Selendroid and create first test script of android application

(http://roadtoautomation.blogspot.ch/2014/05/road-to-setup-selendroid-and-create.html)
* Up and running with: Selendroid

(http://www.ontestautomation.com/up-and-running-with-selendroid/)

一些停止维护的 Android 测试工具

一些几乎没有继续维护的开源 Android 测试工具项目（至少是最近几个月都没有更新的项目）。

Emmagee

Emmagee 是监控指定被测应用在使用过程中占用机器的CPU、内存、流量资源的性能测试小工具。Emmagee 同时还提供非常酷的一些特性，比如定制间隔来收集数据，使用浮动窗口呈现实时进程状态等。

OSChina URL: http://www.oschina.net/p/emmagee

Sirocco

Scirocco（scirocco- webdriver） 是开源的应用自动化测试工具，可以从 Eclipse 访问必要的测试设备。Scirocco 提供自动化的 Android 应用测试功

能，代替手工测试。Scirocco 支持谷歌的 NativeDriver，把 AndroidDriver 作为主要的测试库。Scirocco 包括三个部分：NativeDriver，AndroidDriver，scirocco 插件（一个 Eclipse 插件；可以自动执行 scenario 测试和制作测试报告截图）。

OSChina URL: http://www.oschina.net/p/scirocco

译文链接：http://www.oschina.net/news/56142/open-source-android-testing-tools

原文链接：http://www.softwaretestingmagazine.com/knowledge/open-source-android-testing-tools/

你不需要Bootstrap

译者：腊八粥

有人听到我说对Bootstrap没有好感，是不会感到惊奇的。Four Kitchens【注1】为在各种Drupal活动上的响应式web设计和公司所做培训的其中一个目标，就是为开发者提供他们需要的工具，而不要使用 Bootstrap或其它类似的工具。我希望澄清 我觉得Bootstrap 对于大多数网站是错误的工具 的原因，以及你能够用什么来替代。

1.反设计模式

首先，Bootstrap支持太多的反设计模式。反设计模式是一种看似不错的设计思想，经常被复制，但是对于一个网站来说，通常是糟糕的思想。首先，Bootstrap没有给你一个真正的响应式设计。根据 我们正在把网站建立到手机、平板和桌面上的思想，它应当有4个breakpoint。这与基于内容建立网站、不管用什么设备浏览都要确保站点可用性，是相 反的。关于设备如何不能适应这种离散类别的一个好例子就是三星的Galaxy Mega，它既是一个较大的智能手机，又是一个小平板。设计应该兼顾到这种设备，即使它不是一个明确的平板或智能手机。

基于class的网格系统通常也是一个问题，因为我们正在限制自己。网格应该适应内容本身，而不是对class的结构产生影响。我们不应当使用breakpoint来限制我们做一个12列的布局。我们应该围绕内容设计网站，产生让人尖叫的移动优先的布局。一句话：我们值得拥有更好的。

2.你应当使用自己的设计

Bootstrap网站看起来都一样，它们都用到了相同的前端代码。甚至这些主题使用了很多你或许不需要的样式。不过对于你的网站而言，你应该用尽可能少的CSS来做你的设计。当你从一开始就只能用你自己代码的时候，为什么你应该覆盖一些东西呢?

3.更好的标签

我讨厌过多的class、非语义的class搞乱我的标签。我想让我的网站是干净的、易读的，在我产生标签时尽可能少些阻碍。除了对于干净标签的期望，还有不想使用这种基于class的系统的若干理由。很多CMS，包括Drupal在内，都用自己的观点和方法来输出标签和class，把 Bootstrap贯穿在标签里，这是非常痛苦的。是的，它管用，不过，有很多方法。但是，这比你从头就产生自定义CSS要花费更多的时间和精力。

4.Saas

在我的论点里，这是一个关键点，我通常得到如下反应，“是的，但是我需要制作快速原型，从第一天起我就没有时间自己写CSS。”在Sass社区开始制作可插拔扩展、以支持使用100%的自定义代码之前，刚才说的没错。不仅仅你能用自定义代码，而且从第一天起你就在根据原型编写可发布的代码。Team Sass已经做了大量工作以支持custom grid、media query和极度容易的样式。你可以的，这些工具使之成为可能。

5.Bootstrap的适用范围

我强烈建议不要使用Bootstrap，永远不要提倡在一个线上网站使用。话虽如此，它也不全是不好的，它是一群优秀的工程师和设计师向世界贡献的一些伟大思想。首先，它是学习的一种好方式，可以看看一些有趣的样式和JavaScript是如何被使用的。如果你是前端工作的新手，或者想在一个基础的 层级看看响应式网站设计是如何运行的，那么签出代码。这是学习一些前端例子如何运行的不错的方法。还有，如果你正在开发一个后端管理页面，只在内部使用， 那么Bootstrap就是一个伟大的工具，你不必做全部的设计就拥有了UI块。由于这种代码不需要极好的性能，不管其设计是不是从其它UI元素偷来的， 这都没有关系，Bootstrap就是最好选择。

总结

不要使用Bootstrap。真的，不要用。你可以做得更好，设计得更好，代码写得更好。像Sass之类的CSS预处理器让你得到需要的快速原型，在这一点上，Sass的社区工具就是你的黄油面包。

注1：Four Kitchens正是原文的网站。

译文链接：http://www.labazhou.net/2014/10/you-dont-need-bootstrap/

原文链接：http://fourword.fourkitchens.com/article/you-dont-need-bootstrap

PHP程序员的技术成长规划

作者：黑夜路人

按照了解的很多PHP/LNMP程序员的发展轨迹，结合个人经验体会，抽象出很多程序员对未来的迷漫，特别对技术学习的盲目和慌乱，简单梳理了这个每个阶段PHP程序员的技术要求，来帮助很多PHP程序做对照设定学习成长目标。

本文按照目前主流技术做了一个基本的梳理，整个是假设PHP程序员不是基础非常扎实的情况进行的设定，并且所有设定都非常具体明确清晰，可能会让人觉得不适，请理解仅代表一家之言。（未来技术变化不在讨论范围）

第一阶段：基础阶段（基础PHP程序员）

重点：把LNMP搞熟练（核心是安装配置基本操作）

目标：能够完成基本的LNMP系统安装，简单配置维护；能够做基本的简单系统的PHP开发；能够在PHP中型系统中支持某个PHP功能模块的开发。

时间：完成本阶段的时间因人而异，有的成长快半年一年就过了，成长慢的两三年也有。

1.Linux：

基本命令、操作、启动、基本服务配置（包括rpm安装文件，各种服务配置等）；会写简单的shell脚本和awk/sed 脚本命令等。

2.Nginx：

做到能够安装配置nginx+php，知道基本的nginx核心配置选项，知道 server/fastcgi_pass/access_log 等基础配置，目标是能够让nginx +php_fpm顺利工作。

3.MySQL：

会自己搭建mysql，知道基本的mysql配置选项；知道innodb和myisam的区别，知道针对InnoDB和MyISAM两个引擎的不同配置选 项；知道基本的两个引擎的差异和选择上面的区别；能够纯手工编译搭建一个MySQL数据库并且配置好编码等正常稳定运行；核心主旨是能够搭建一个可运行的MySQL数据库。

4.PHP：

基本语法数组、字符串、数据库、XML、Socket、GD/ImageMgk图片处理等等；熟悉各种跟MySQL操作链接的api（mysql /mysqli/PDO)，知道各种编码问题的解决；知道常规熟练使用的PHP框架（ThinkPHP、Zendframework、Yii、Yaf 等）；了解基本MVC的运行机制和为什么这么做，稍微知道不同的PHP框架之间的区别；能够快速学习一个MVC框架。能够知道开发工程中的文件目录组织， 有基本的良好的代码结构和风格，能够完成小系统的开发和中型系统中某个模块的开发工作。

5.前端：

如果条件时间允许，可以适当学习下 HTML/CSS/JS 等相关知识，知道什么web标准，div+css的web/wap页面模式，知道HTML5和 HTML4的区别；了解一些基本的前端只是和JS框架（jQuery之类的）；了解一些基本的JavaScript编程知识；（本项不是必须项，如果有时 间，稍微了解一下是可以的，不过不建议作为重点，除非个人有强烈兴趣）

6.系统设计：

能够完成小型系统的基本设计，包括简单的数据库设计，能够完成基本的：浏览器 -> Nginx+PHP -> 数据库 架构的设计开发工作；能够支撑每天几十万到数百万流量网站的开发维护工作；

第二阶段：提高阶段 （中级PHP程序员）

重点：提高针对LNMP的技能，能够更全面的对LNMP有熟练的应用。

目标：能够随时随地搭建好LNMP环境，快速完成常规配置；能够追查解决大部分遇到的开发和线上环境的问题；能够独立承担中型系统的构架和开发工作；能够在大型系统中承担某个中型模块的开发工作；

1. Linux:

在第一阶段的基础上面，能够流畅的使用Shell脚本来完成很多自动化的工作；awk/sed/perl 也操作的不错，能够完成很多文本处理和数据统计等工作；基本能够安装大部分非特殊的Linux程序（包括各种库、包、第三方依赖等等，比如MongoDB/Redis/Sphinx/Luncene /SVN之类的）；了解基本的Linux服务，知道如何查看Linux的性能指标数据，知道基本的Linux下面的问题跟踪等。

2. Nginx:

在第一阶段的基础上面，了解复杂一些的Nginx配置；包括 多核配置、events、proxy_pass，sendfile/tcp_*配置，知道超时 等相关配置和性能影响；知道nginx除了web server，还能够承担代理服务器、反向静态服务器等配置；知道基本的nginx配置调优；知道如何 配置权限、编译一个nginx扩展到nginx；知道基本的nginx运行原理（master/worker机制，epoll），知道为什么nginx性 能比apache性能好等知识；

3. MySQL/MongoDB：

在第一阶段的基础上面，在MySQL开发方面，掌握很多小技巧，包括常规SQL优化（group by/order by/rand优化等）；除了能够搭建 MySQL，还能够冷热备份MySQL数据，还知道影响innodb/myisam性能的配置选项（比如key_buffer/query_cache / sort_buffer /innodb_buffer_pool_size/ innodb_flush_log_at_trx_commit等），也知 道这些选项配置成为多少值合适；另外也了解一些特殊的配置选项，比如 知道如何搭建mysql主从同步的环境，知道各个binlog_format的区 别；知道MySQL的性能追查，包括slow_log/explain等，还能够知道基本的索引建立处理等知识；原理方面了

解基本的MySQL的架构 （Server+存储引擎），知道基本的InnoDB/MyISAM索引存储结构和不同（聚簇索引，B树）；知道基本的InnoDB事务处理机制；了解大 部分MySQL异常情况的处理方案（或者知道哪儿找到处理方案）。条件允许的情况，建议了解一下NoSQL的代表MongoDB数据库，顺便对比跟 MySQL的差别，同事能够在合适的应用场景安全谨慎的使用MongoDB，知道基本的PHP与MongoDB的结合开发。

4. Redis/Memcached：

在大部分中型系统里面一定会涉及到缓存处理，所以一定要了解基本的缓存；知道Memcached和Redis的异同和应用场景，能够独立安 装 Redis/Memcached，了解Memcahed的一些基本特性和限制，比如最大的value值，知道PHP跟他们的使用结合；Redis了解 基本工作原理和使用，了解常规的数据类型，知道什么场景应用什么类型，了解Redis的事务等等。原理部分，能够大概了解Memcached的内存结构 （slab机制），redis就了解常用数据类型底层实现存储结构（SDS/链表/SkipList/HashTable）等等，顺便了解一下Redis 的事务、RDB、AOF等机制更好

5. PHP：

除了第一阶段的能力，安装配置方面能够随意安装PHP和各种第三方扩展的编译安装配置；了解php-fpm的大部分配置选项和含义（如max_requests/max_children/request_terminate_timeout之类的影响性能的配置），知道mod_php /fastcgi的区别；在PHP方面已经能够熟练各种基础技术，还包括各种深入些的PHP，包括对PHP面向对象的深入理解/SPL/语法层面的特殊特 性比如反射之类的；在框架方面已经阅读过最少一个以上常规PHP MVC框架的代码了，知道基本PHP框架内部实现机制和设计思想；在PHP开发中已经能 够熟练使用常规的设计模式来应用开发（抽象工厂/单例/观察者/命令链/策略/适配器 等模式）；建议开发自己的PHP MVC框架来充分让开发自由化， 让自己深入理解MVC模式，也让自己能够在业务项目开发里快速升级；熟悉PHP的各种代码优化方法，熟悉大部分PHP安全方面问题的解决处理；熟悉基本的 PHP执行的机制原理（Zend引擎/扩展基本工作机制）；

6. C/C++：

开始涉猎一定的C/C++语言，能够写基本的C/C++代码，对基本的C/C++语法熟悉（指针、数组操作、字符串、常规标准API）和数据结构（链表、 树、哈希、队列）有一定的熟悉下；对Linux下面的C语言开发有基本的了解概念，会简单的makefile文件编写，能够使用简单的GCC/GDB的程序编译简单调试工作；对基本的网络编程有大概了解。（本项是为了向更高层次打下基础）

7. 前端：

在第一阶段的基础上面，熟悉基本的HTTP协议（协议代码200/300/400/500，基本的HTTP交互头）；条件允许，可以在深入写出稍微优雅的 HTML+CSS+JavaScript，或者能够大致简单使用某些前端框架（jQuery/YUI/ExtJS/RequireJS /BootStrap之类）；如果条件允许，可以深入学习JavaScript编程，比如闭包机制、DOM处理；再深入些可以读读jQuery源码做深入 学习。（本项不做重点学习，除非对前端有兴趣）

8. 系统设计：

能够设计大部分中型系统的网站架构、数据库、基本PHP框架选型；性能测试排查处理等；能够完成类似：浏览 器 -> CDN(Squid) -> Nginx+PHP -> 缓存 -> 数据库 结构网站的基本设计开发维护；能够支撑每天数百万到千万流量基本网站的开发维护工作；

第三阶段：高级阶段 （高级PHP程序员）

重点：除了基本的LNMP程序，还能够在某个方向或领域有深入学习。（纵深维度发展）

目标：除了能够完成基本的PHP业务开发，还能够解决大部分深入复杂的技术问题，并且可以独立设计完成中大型的系统设计和开发工作；自己能够独立hold深入某个技术方向，在这块比较专业。（比如在MySQL、Nginx、PHP、Redis等等任一方向深入研究）

1. Linux：

除了第二阶段的能力，在Linux下面除了常规的操作和性能监控跟踪，还能够使用很多高级复杂的命令完成工作（watch/tcpdump/starce /ldd/ar 等)；在shell脚本方面，已经能够编写比较复杂的shell脚本（超过500行）来协助完成很多包括备份、自动化处理、监控等工作的 shell；对awk/sed/perl 等应用已经如火纯青，能够随意操作控制处理文本统计分析各种复杂格式的数据；对Linux内部机制有一些了解， 对内核模块加载，启动错误处理等等有个基本的处理；同时对一些其他相关的东西也了解，比如NFS、磁盘管理等等；

2. Nginx:

在第二阶段的基础上面，已经能够把Nginx操作的很熟练，能够对Nginx进行更深入的运维工作，比如监控、性能优化，复杂问题处理等等；看个人兴趣，更 多方面可以考虑侧重在关于Nginx工作原理部分的深入学习，主要表现在阅读源码开始，比如具体的master/worker工作机制，Nginx内部的事件处理，内存管理等等；同时可以学习Nginx扩展的开发，可以定制一些自己私有的扩展；同时可以对Nginx+Lua有一定程度的了解，看看是否可以结合应用出更好模式；这个阶段的要求是对Nginx原理的深入理解，可以考虑成为Nginx方向的深入专业者。

3. MySQL/MongoDB：

在第二阶段的基础上面，在MySQL应用方面，除了之前的基本SQL优化，还能够在完成一些复杂操作，比如大批量数据的导入导出，线上大批量数据的更改表结 构或者增删索引字段等等高危操作；除了安装配置，已经能够处理更多复杂的MySQL的问题，比如各种问题的追查，主从同步延迟问题的解决、跨机房同步数据 方案、MySQL高可用架构等都有涉及了解；对MySQL应用层面，对MySQL的核心关键技术比较熟悉，比如事务机制（隔离级别、锁等）、对触发器、分 区等技术有一定了解和应用；对MySQL性能方面，有包括磁盘优化（SAS迁移到SSD）、服务器优化（内存、服务器本身配置）、除了二阶段的其他核心性 能优化选项（innodb_log_buffer_size/back_log/table_open_cache/thread_cache_size/innodb_lock_wait_timeout等）、连接池软件选择应用，对 show *（show status/show profile）类 的操作语句有深入了解，能够完成

大部分的性能问题追查；MySQL备份技术的深入熟悉，包括灾备还原、对Binlog的深入理解，冷热备份，多IDC备份 等；在MySQL原理方面，有更多了解，比如对MySQL的工作机制开始阅读部分源码，比如对主从同步（复制）技术的源码学习，或者对某个存储引擎 （MyISAM/Innodb/TokuDB）等等的源码学习理解，如果条件允许，可以参考CSV引擎开发自己简单的存储引擎来保存一些数据，增强对 MySQL的理解；在这个过程，如果自己有兴趣，也可以考虑往DBA方向发展。MongoDB层面，可以考虑比如说在写少读多的情况开始在线上应用 MongoDB，或者是做一些线上的数据分析处理的操作，具体场景可以按照工作来，不过核心是要更好的深入理解RMDBS和NoSQL的不同场景下面的应 用，如果条件或者兴趣允许，可以开始深入学习一下MongoDB的工作机制。

4. Redis/Memcached：

在第二阶段的基础上面，能够更深入的应用和学习。因为Memcached不是特别复杂，建议可以把源码进行阅读，特别是内存管理部分，方便深入理解；Redis部分，可以多做一些复杂的数据结构的应用（zset来做排行榜排序操作/事务处理用来保证原子性在秒杀类场景应用之类的使用操作）；多涉及 aof等同步机制的学习应用，设计一个高可用的Redis应用架构和集群；建议可以深入的学习一下Redis的源码，把在第二阶段积累的知识都可以应用 上，特别可以阅读一下包括核心事件管理、内存管理、内部核心数据结构等充分学习了解一下。如果兴趣允许，可以成为一个Redis方面非常专业的使用者。

5. PHP：

作为基础核心技能，我们在第二阶段的基础上面，需要有更深入的学习和应用。从基本代码应用上面来说，能够解决在PHP开发中遇到95%的问题，了解大部分 PHP的技巧；对大部分的PHP框架能够迅速在一天内上手使用，并且了解各个主流PHP框架的优缺点，能够迅速方便项目开发中做技术选型；在配置方面，除 了常规第二阶段会的知识，会了解一些比较偏门的配置选项（php auto_prepend_file/auto_append_file），包括扩展中的一些复杂高级配置和原理（比如memcached扩展配置中的memcache.hash_strategy、apc扩展配置中的apc.mmap_file_mask/apc.slam_defense/apc.file_update_protection之类的）；对php的 工作机制比较了解，包括php-fpm工作机制（比如php-fpm

在不同配置机器下面开启进程数量计算以及原理），对zend引擎有基本熟悉 （vm/gc/stream处理），阅读过基本的PHP内核源码（或者阅读过相关文章），对PHP内部机制的大部分核心数据结构（基础类型/Array /Object）实现有了解，对于核心基础结构（zval/hashtable/gc）有深入学习了解；能够进行基本的PHP扩展开发，了解一些扩展开发 的中高级知识（minit/rinit等），熟悉php跟apache/nginx不同的通信交互方式细节（mod_php/fastcgi）；除了开发 PHP扩展，可以考虑学习开发Zend扩展，从更底层去了解PHP。

6. C/C++：

在第二阶段基础上面，能够在C/C++语言方面有更深入的学习了解，能够完成中小型C/C++系统的开发工作；除了基本第二阶段的基础C/C++语法和数据 结构，也能够学习一些特殊数据结构（b-tree/rb-tree/skiplist/lsm-tree/trie-tree等）方便在特殊工作中需求； 在系统编程方面，熟悉多进程、多线程编程；多进程情况下面了解大部分多进程之间的通信方式，能够灵活选择通信方式（共享内存/信号量/管道等）；多线程编 程能够良好的解决锁冲突问题，并且能够进行多线程程序的开发调试工作；同时对网络编程比较熟悉，了解多进程模型/多线程模型/异步网络IO模型的差别和选 型，熟悉不同异步网络IO模型的原理和差异（select/poll/epoll/iocp等），并且熟悉常见的异步框架（ACE/ICE/libev /libevent/libuv/Boost.ASIO等）和使用，如果闲暇也可以看看一些国产自己开发的库（比如muduo）；同时能够设计好的高并发 程序架构（leader-follow/master-worker等）；了解大部分C/C++后端Server开发中的问题（内存管理、日志打印、高并 发、前后端通信协议、服务监控），知道各个后端服务RPC通信问题（struct/http/thirft/protobuf等）；能够更熟络的使用 GCC和GDB来开发编译调试程序，在线上程序core掉后能够迅速追查跟踪解决问题；通用模块开发方面，可以积累或者开发一些通用的工具或库（比如异步 网络框架、日志库、内存池、线程池等），不过开发后是否应用要谨慎，省的埋坑去追bug；

7. 前端：

深入了解HTTP协议（包括各个细致协议特殊协议代码和背后原因，比如302静态文件缓存了，502是nginx后面php挂了之类的）；除了之前的前端方 面的各种框架应用整合能力，前端方面的学习如果有兴趣可以更深入，

表现形式是，可以自己开发一些类似jQuery的前端框架，或者开发一个富文本编辑器之 类的比较琐碎考验JavaScript功力；

8. 其他领域语言学习：

在基础的PHP/C/C++语言方面有基本积累，建议在当前阶段可以尝试学习不同的编程语言，看个人兴趣爱好，脚本类语言可以学学 Python /Ruby 之类的，函数式编程语言可以试试 Lisp/Haskell/Scala/Erlang 之类的，静态语言可以试试 Java /Golang，数据统计分析可以了解了解R语言，如果想换个视角做后端业务，可以试试 Node.js还有前面提到的跟Nginx结合的 Nginx_Lua等。学习不同的语言主要是提升自己的视野和解决问题手段的差异，比如会了解除了进程/线程，还有轻量级协程；比如在跨机器通信场景下 面，Erlang的解决方案简单的惊人；比如在不想选择C/C++的情况下，还有类似高效的Erlang/Golang可用等等；主要是提升视野。

9. 其他专业方向学习：

在本阶段里面，会除了基本的LNMP技能之外，会考虑一些其他领域知识的学习，这些都是可以的，看个人兴趣和长期的目标方向。目前情况能够选择的领域比较 多，比如、云计算（分布式存储、分布式计算、虚拟机等），机器学习（数据挖掘、模式识别等，应用到统计、个性化推荐），自然语言处理（中文分词等），搜索 引擎技术、图形图像、语音识别等等。除了这些高大上的，也有很多偏工程方面可以学习的地方，比如高性能系统、移动开发（Android/IOS）、计算机 安全、嵌入式系统、硬件等方向。

10. 系统设计：

系统设计在第二阶段的基础之上，能够应用掌握的经验技能，设计出比较复杂的中大型系统，能够解决大部分线上的各种复杂系统的问题，完成类似 浏览 器 -> CDN -> 负载均衡 ->接入层 -> Nginx+PHP -> 业务缓存 -> 数据 库 -> 各路复杂后端RPC交互（存储后端、逻辑后端、反作弊后端、外部服务） -> 更多后端 酱紫的复杂业务；能够支撑每天数千万到数 亿流量网站的正常开发维护工作。

第四阶段：架构阶段 （架构师）

ps: 暂时不展开讨论，等下次专门撰文来描述补充本部分内容

第五阶段：专家阶段（方向领域专家）

ps：高大上，这块不展开讨论 ^_^

第六阶段：科学家阶段

ps：高大上，这块不展开讨论 ^_^

原文链接：http://blog.csdn.net/heiyeshuwu/article/details/40098043

阿里云监控体系现状概览

作者：杨赛

本文根据InfoQ中文站跟阿里云产品技术部产品总监马劲的在2014年10月初的一次电话交流整理而成，褚霸对本采访内容亦有所贡献。在本次沟通中，马劲对阿里云监控体系的现状进行了简单介绍，涉及到监控的覆盖面、监控粒度、故障识别、OpenAPI的开放进度等方面。

嘉宾简介

马劲，花名竹蜓，来自阿里云产品技术部，是阿里云四大主题（云服务器，存储和多媒体、数据平台、中间件）的产品总监之一，负责阿里云所有中间件产品 管理，目前负责管理的产品有OCS（缓存）、MQS（消息队列）、ACE（云引擎）、ESS（弹性扩展）、PTS（性能测试）、Open Search（开放搜索）、ONS（开放消息服务）、云监控。竹蜓之前在IBM有13年工作经历，曾经担任云计算软件全球产品总监、大中华区软件VIP客户服务总监、大中华区软件培训负责人、攻城狮、程序猿等多个职位。

背景概述

监控体系是云计算基础架构最重要的组成部分之一。阿里云监控体系有两个视角：运维自己的集群监控体系，由技术保障部主导；以及用户视角的监控产品 （如云监控服务），由产品技术部主导。运维的工作着眼于掌握每个服务的可用性、可靠性数据，提升发现问题解决问题的速度；客户的需求则是可以方便的看到自 己阿里云资源的状态，包括资源的稳定情况和资源消耗情况等，以及客户基于阿里云的应用的状态，例如应用是否可用，性能如何。

当然，无论是何种角度的监控，底层基础架构是共享的。

过去一年主要完成的相关工作包括：

- 为阿里云的各个产品逐步建立全链路监控，完成对所有服务各个模块端到端的数据采集（运维视角）
- 在2014年4月开始“云监控”产品的公测，目前包含站点监控、ECS监控以及自定义监控。ECS的CPU、内存、IO、存储等资源的状态，现在所有人都可以通过API项获取（用户视角）
- RDS、SLB在“云监控”上的接入正在实现当中（用户视角）

RDS的监控

褚霸：RDS全链路监控现在基本都做完了。RDS较早做到全链路监控，是因为用户对数据库的QPS和RT变化非常敏感，倒逼着我们对系统更深入把控。

这就要求我们能够从网络（交换机）、操作系统、LVS、中间层、数据库整个链路能够从用户的视角收集到详尽的数据，通过流计算平台聚合、对比来看到异常，先于用户发现变坏的趋势，在更短的时间内做出响应。

全链路是个麻绳，需要整个链条上的每个部件都暴露出足够多的信息（特别是用户触发的行为动作），透过这个麻绳串起来。比如说RDS中间层统计数据在 内存中按照树状组织的，基本上所有的内部模块都有详细的运行状态，日志和诊断代码在中间层中占比达到30%。这些状态都在内存中，抓取统计数据的代价是恒 定的，精细到每个用户的链接，再聚合成用户级别，主机级别，集群级别在监控系统界面上关联展现，指标超过150个，基本可以做到白盒，对系统的运行了如指 掌。

监控粒度

ECS上的CPU争抢情况已经按秒级采集数据，而对于IO访问的访问请求监控是更细粒度，统计到每个IO访问的响应延时。

监控的目的提供稳定的服务，在出了问题以后能尽快处置，即使做到十毫秒级，如果还是基于事件，问题报警，也是事后诸葛亮。监控希望能做到事前分析与 预测，所谓后发先至，避免发生影响服务的事件，这本身是一个IT数据的大数据应用的课题，例如我们正在分析VM的CPU消耗周期变化尽可能把CPU密集的 VM均匀分布到不同的物理机器上，同时正在开发动态热点迁移技术进一步提高用户体验。

做到秒级不是目标，做到主动预测、主动干预化解问题，避免服务对外不可用，才是我们的目标。

故障预测

基于全链路的监控与分析平台，我们对每一次的故障进行review，将故障原因的相关指征提取出来，形成预警方法。有些故障是由软件更新的bug触 发，不过bug触发的问题如果能够提取为指征，也可以回归到预警系统。另外，通过异常分析也可能找出可能未知的问题，报警让人来分析。

全链路监控与分析平台现在在RDS上得到应用已经取得不错的效果，ECS、SLB和CDN等正在应用该平台。

弹性计算服务

阿里云正在做Elastic Scaling Service弹性计算服务，原计划8月发布，但是觉得还达不到公测的质量水平标准，所以有所延迟，预计近期（10月）会推出邀请测试。

第三方监控服务

其实公测阶段的云监控产品目前还不太完善。比如ECS实例监控，现在还需要客户手动下载安装Agent，这对于ECS服务器数量多的客户是工作量很大的，可以做得更加自动化。

阿里云的云监控会提供更多服务，同时也欢迎第三方能够针对阿里云开发一些高级的监控。目前市场上已经有客户自己安装了第三方监控可以在阿里云上使 用，包括商业监控软件和开源监控软件对阿里云资源的监控，也有客户自己订阅了监控SaaS服务来监控他们跑在阿里云上的应用，包括一些国外的SaaS监控 （如New Relic）。

阿里云的云监控提供OpenAPI，目前已经在针对小部分可信用户进行内测。内测资格目前只有经过单独审批才能拿到，主要针对企业客户。因为API 涉及到权限、流量等安全因素，所以这方面会非常谨慎的逐步公开，一方面要借助内测用户的尖锐批评来改进，减少bug、提升用户体验，另一方面也需要把文档 更加完善起来，具体的时间表尚未确定。

总结

阿里云是非常技术的产品，但归根结底是为了让客户用的爽，解决客户的问题。阿里云总裁菲青经常带阿里云的管理者去跟客户沟通，管理者也被鼓励尽量带 着一线员工出去了解客户。现在每次阿里云发新的feature之前，产品经理都会先把demo发到客户群里，这样在上线之前就能收集到部分反馈。今年9月 初，阿里云管理者大会上搞了“火线24小时”的活动，全员自由组了几十个队伍去研究客户工单，在24小时内针对工单内容设计解决方案，最后选拔出来的8支 队伍提供的方案在大会上讲演之后立刻往下迭代，不需要立项、审批、排期。这样的活动还会不定期的搞下去，只要客户有问题，阿里云就有动力持续的努力解决。

做阿里云，技术上固然有很多挑战，但最大的挑战还是对客户的理解。因为业务是很丰富的，比如客户提一个要查看某一个监控项比如缓存命中率的需求，我 技术上实现不难，但做出来的东西未必是客户想要的，客户可能查看数据主要的目的是找到应用出了什么问题，具体观察的时间，观察到以后如何处理。唯有真的去 客户那里和客户交流，深入了解客户使用的场景，了解客户的痛点，才能做出真正满足客户需求的服务。

工单，论坛都是是很重要的产品改进输入，我们的产品经理和管理者会认真看工单，不断转化为产品改进。恳请大家把您的问题告诉我们，“向客户学习，陪伴客户成长”是我们的理念。

原文链接：http://www.infoq.com/cn/news/2014/10/aliyun-monitor-system-overview

再看知名应用背后的第三方开源项目

作者：suiling

知名应用程序的设计和技术一直都是开发者需要学习的，同样这些应用所使用的开源框架也是不可忽视的一部分。此前《iOS第三方开源库的吐槽和备忘》中作者ibireme列举了国内多款知名应用所使用的开源框架，并对其中一些框架进行了分析，同样国外开发者@iOSCowboy也在博客中给我们列出了国外多款知名应用使用的开源框架。另外txx's blog中详细介绍了Facebook Paper使用的第三方库。

Instagram

AFNetworking: 适用于iOS和OS X的网络框架。

Appirater: 提醒用户打分。

ASIHTTPRequest：简单使用CFNetwork API封装进行HTTP网络请求，用Objective-C编写，可应用在Mac OSX和iOS开发中。

CocoaHTTPServer: 用于Mac OS X和iOS应用程序的轻量级、可嵌入的HTTP服务器框架。

Cocoa Lumberjack:适用于Mac和iOS的日志框架，集简单、快速、强大以及灵活于一身。

MBProgressHUD: 用多种样式展示半透明的HUD，并带有指示器和标签，自定义功能强大。

PLCrashReporter (Github mirror): 进程内崩溃报告框架。

QSUtilities: 实用工具、控件以及其他辅助类的集合。

SocketRocket: Objective-C WebSocket客户端库。
https://github.com/square/SocketRocket

XBImageFilters:允许实时过滤摄像头拍摄的照片，使用OpenGL ES 2 来快速处理各种图片效果。

Foursquare

Facebook SDK for iOS: 集成Facebook,构建强大的社交app。

FSNetworking: Foursquare iOS网络库。

kingpin: MapKit/MKAnnotation pin 聚合库，主要用来在地图上面添加锚点。

AFNetworking:适用于iOS和OS X的网络框架。

SKBounceAnimation: CAKeyframeAnimation子类，可快速简单地设置弹动的数量，开始和结束的值，以及创建动画。

DB5: 通过Plist配置文件。

LinkedIn

BlocksKit: blocks工具包。

SDWebImage: 提供一个UIImageVIew类以支持远程加载网络图片。具有缓存管理、异步图片下载等功能，支持GIF动画，使用GCD和ARC。

DTCOreText:文字效果代码类库。在UITextView上实现丰富的文字效果，比如文字大小、颜色、字体、下划线，链接，给文字加上图片、视频，文字任意间距等等。实现类似于CSS网页的文字效果。

Shazam

AudioStreamer:Mac OS X和i-Phone上适用的流媒体音频播放器，可播放来自网络上的音乐。.

ColorArt: iTunes 11风格的颜色匹配代码。

objc-geohash: Objective-C GeoHash库，通过经纬度获得哈希表。

FormatterKit: 收集了精心构思的NSFormatter子类。

UIView+Glow: UIView的一个类别，可添加对制作发光视图的支持，以突出屏幕上重要的部分，方便用户与之进行交互。

WEbViewJavascriptBridge: 在使用UIWeb-View时，它优雅地实现了JS与ios 的ObjC 原生代码之间的互调，支持消息发送、接收、消息处理器的注册与调用以及设置消息处理的回调。

Skype

AFNetworking: 适用于iOS和OS X的网络框架。

Hockey SDK: HockeyApp service官方iOS SDK。

PLCrashReporter (Github mirror): 进程内的崩溃报告框架。

TTTAttributedLabel是一个文字视图开源组件，是UILabel的替代元件，可以以简单的方式展现渲染的属性字符串。另外，还支持链接植入，不管是手动还是使用UIDataDetectorTypes自动把电话号码、事件、地址以及其他信息变成链接。

SDWebImage: 提供一个UIImageVIew类以支持远程加载网络图片。具有缓存管理、异步图片下载等功能，支持GIF动画，使用GCD和ARC。

Cocoa Lumberjack: 适用于Mac和iOS的日志框架，集简单、快速、强大以及灵活于一身。

MWPhotoBrowser: 一个简单的带有栅格视图的iOS照片浏览器，可添加标题和选择多个图片。照片浏览器效果类似iOS原生的照片应用,可显示来自手机的图片或者是网络图片，也可自动从网络下载图片并进行缓存，还可图片进行缩放等。

BlocksKit: Objective-C blocks工具包。

Spotify

FMDB: SQLite API封装库。

MAObjCRuntime:将运行时API封装成ObjC。

Nu: 编程语言。

PLCrashReporter (Github mirror):进程内崩溃报告框架。

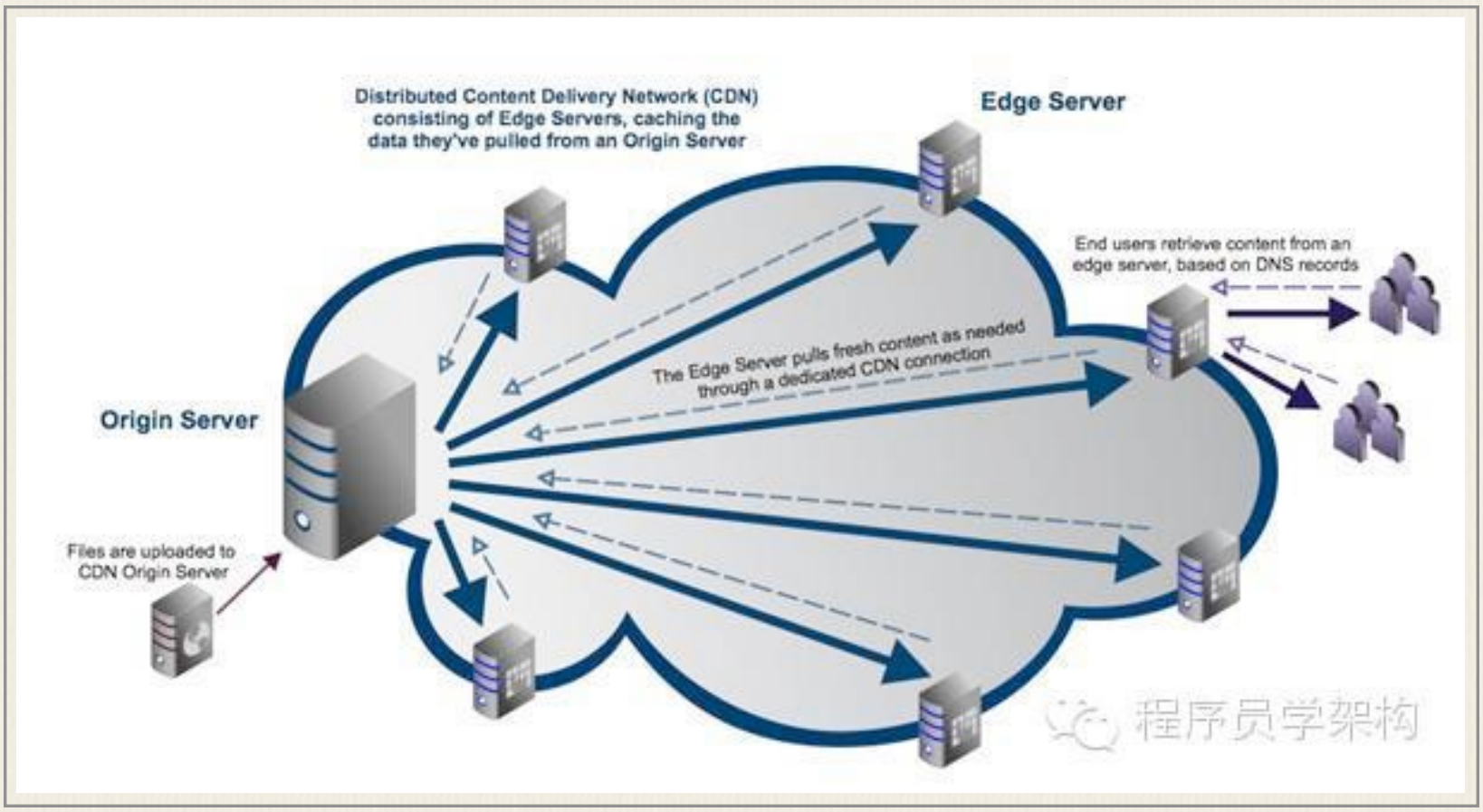
SBJSON:Objective-C 实现的一个严格的JSON 解析器和生成器。

原文链接：http://www.cocoachina.com/ios/20141017/9955.html

分布式还是混合式? 谈CDN架构对服务质量的影响

译者：mathew

传统分布式模型



通 常，内容分发网络(CDN)采用分布式模型。在这种模型里， 用户的文件存放在一个源服务器上，并且由大量边缘服务器负责分发这些文件。这些边缘服务器的磁盘空间比较小，所以大多数的文件被放在内存中。因此，当一个 文件传输至终端用户之前，如果该文件在边缘的CDN缓存服务器上不存在，那么这些服务器会先去原始服务器请求该文件或流。

这种分布式模型起 源于20世纪末，也就是在那时全球第一个CDN开始出现。当时在2000年的时候，CDN的主要挑战是从互联网服务提供商(ISP)的内部站点 (POPs)去传输网页内容。每一个接入POP的终端用户都会快速获

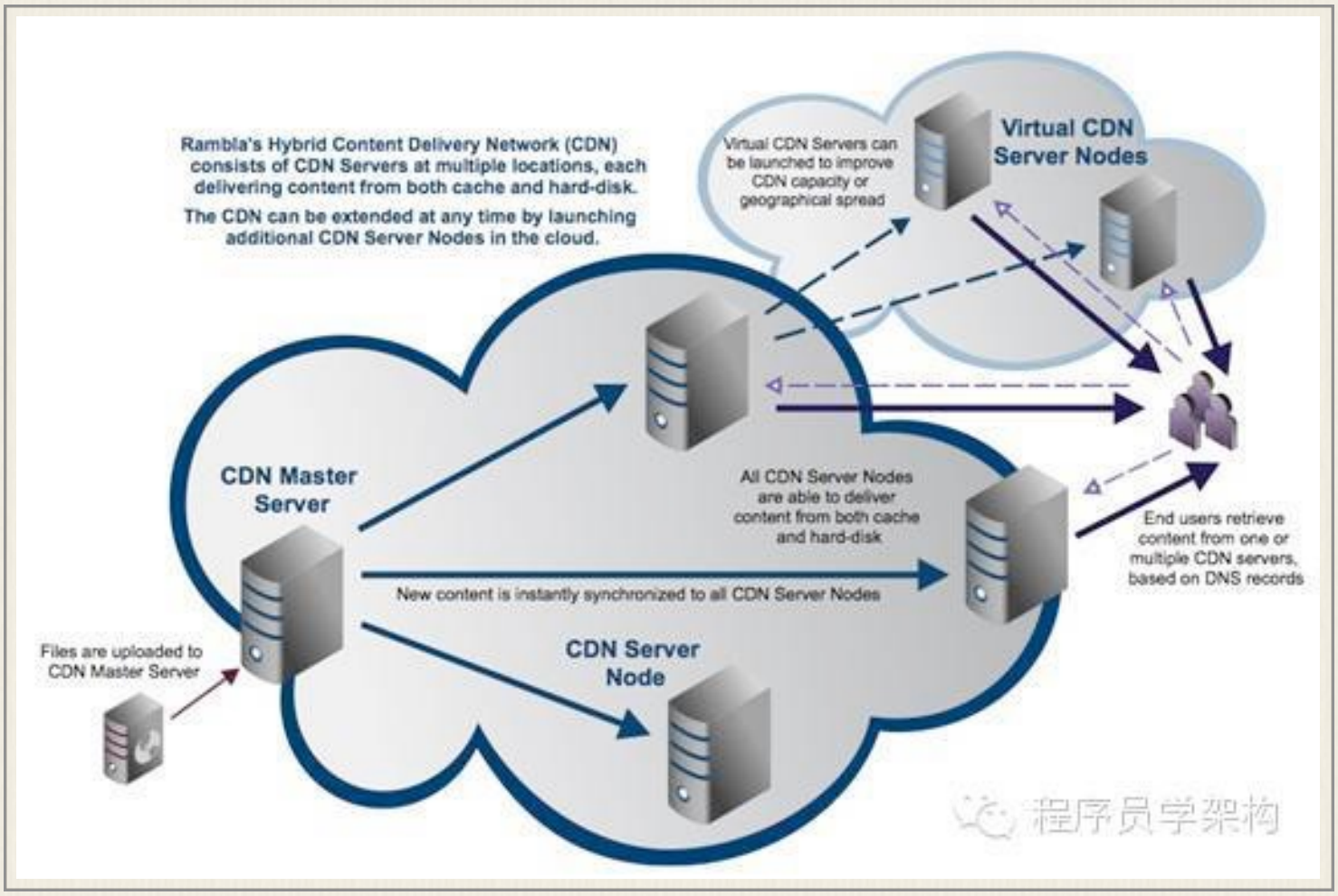
得响应内容，而不是CDN首先通过网络--当时依然很慢--来访问源服务器；因为在这 种分布式CDN架构里，边缘服务器部署是部署在ISP内部用来缓存从源服务器拉下来的内容的。通过这种方式，CDN可以很容易把受欢迎的内容发送给大量的 互联网用户。

多媒体趋势暴露了传统分布式模型的缺点

然 而，随着互联网带宽和连接的爆炸性增长，传统的分布式架构越来越不明显。例如，欧洲有一个非常密集的互联网干线用来连接公共的网络交换点和私人的p2p网 络，而且绝大多说终端用户都可以转让自己的入网许可。在这样的环境下，为每个POP网络部署边缘服务器就不是很有意义，尽管CDN的可扩展性和负载均衡的 能力变得非常重要。

同时，在线多媒体消费的变化揭示了CDN传统分布式结构的局限性。由于多媒体文件的并发请求数在以几何形式增长，这导致 了流媒体服务、多媒体广告、用户制作的视频以及在线文档等文件量也在不断增多。此外，每个视频必须保证在不同的设备和平台上可用使用不同的格式和分辨率来 观看。同时，由于重放设备和终端用户带宽的增强，视频文件的平均大小也在不断增大。所有这些因素使得从边缘服务器缓存分发多媒体内容变得更加困难。为了满 足需求，磁盘空间有限的边缘服务器将不断地刷新其部分缓存，为当前必须要从源服务器加载的新文件腾出空间。这对传统的CDN分布式架构来说，导致了大量的 开销： 越来越多的边缘服务器需要提供相同数量的数据、大量的内部数据需要通过越来越大的连接管道从源服务器发送到边缘服务器。更要紧的是，对最终用户来说，这 可能会导致波动延迟和较低的服务质量(QoS) 。

RAMBLA 混合式CDN：一个完美的匹配多媒体传输架构



为了解决这些问题，当Rambla CDN在2005年被重新设计时，我们决定设计自己的混合式CDN架构。在这种混合式设计结构中，边缘服务器和源服务器之间没有区别，并且所有的服务器进行了优化以便从他们各自的缓存和磁盘上分发内容。这使得CDN更容易适应如分发多媒体数据的需求，同时使用相对少量的高端服务器。

在我们的混合式CDN架构中，每个CDN服务器都可以扮演一个源服务器的角色：提供存储和分发数据到其他的CDN服务器。每个CDN服务器也可以扮演一个边缘服务器的角色：通过专用的存储区域网络(SAN),直接从其缓存分发频繁被访问的内容，从磁盘分发很少被访问的内容。受益于当前的互联网基础架构，我们的CDN服务器不再需要部署到每个ISP网络内，只要实现所有的CDN服务器都有一个快速的、高带宽连接到主网络和ISPs(通过对等协议)，这样就能满足全欧洲覆盖并能保证较快的下载速度以及为所有最终用户提供一个可靠的连接。

自推出 RAMBLA CDN 以来，我们一直在使用最新的技术来优化这种混合式CDN结构。尤其是，通过使用动态的发布集成的云实例，使得我们的CDN结构拥有了可即时扩展的能力，从而使得从性能上和地理上扩展CDN都成为可能。

更进一步：使用最高质量的服务分发多媒体数据

我们最新的研发方向,与IWT(一家科学技术创新机构)合资，旨在提供一个最佳的和稳定的服务质量(QoS)。超 过一定水平，Qos很难去衡量，因为它和每个流或者下载都不同。例如，如果边缘服务器的缓存中有视频文件，那么将对终端用户观看视频的体验产生重大影响。 因此,很多CDN的只是着重于实现足够的QoS,即一个可满足绝大多数终端用户的满意度即可。虽然这种方法可能足以满足传统的在线视频平台，但这不能满足 新一代的增值服务如流式的想用户应用程序传输音频和视频或者机顶盒，也就是所说的OTT服务。

我们混合式的Rambla CDN被设计成通过平等高速以及可靠的传输所有内容来提供高质量的Qos。这就是我们新的SkyWay软件。这种分布式后端组件简 化所有硬盘访问-指向一个专用SAN(存储区域网络)-来确保随机读操作可以以更快的速度完成，此外，它还持续性的监控CDN服务器和CDN服务器上的资 源，保证不论是缓存中还是硬盘中都有足够的资源来供CDN优化传输使用。为了达到这一目标，SkyWay可以动态推迟执行时间不是很重的进程、限制其他进 程资源的有效性、分配任务到其他服务器、释放并重新创建已有资源。通过这种方式，SkyWay保证服务质量在所有情况下任然最优。

1. 本文由mathew翻译

2. 本文译自rambla.eu文章distributed or hybrid:how cdns architecture affects your quality service

3. 本文出自：程序员学架构

从Apache Storm学到的经验教训

译者：伍昆

在Storm升级为Apache基金会顶级项目后，Storm主工程师Nathan Marz就Storm的由来和项目心得撰写了本文。Nathan Marz表示，Storm的成功离不开开源，打造一个成功的项目不仅是能帮助人们解决实际问题，还必需兼顾文档，推广，支持社区等方方面面。

译文如下：

Storm的由来

首先，任何成功的项目需要做到两点：

1. 成功解决了一个实际问题

2. 能够说服大量的用户使用你的项目，证明你的项目是他们的最佳解决方案。

我想，第二点是开发者在项目工作中普遍忽略的一点，但其实这关系到项目的成败，我希望下述的Storm历史能够使大家对此引起足够的重视。

Storm的前身是BackYype， 用来帮助商家分析和研究他们的产品在社交网络中的影响力，可以处理历史或实时的数据。使用的方法是标准队列和线程(worker)方法，但是这个方案不太 令人满意。首先它并不友好，所有的队列和线程都必须确保可用，这样相对来说显得冗长。大部分逻辑处理用于进行收发和序列/反序列消息，一个应用的某个逻辑 将会遍及所有线程。

在2010年12月，我想到了“流(Stream)”的分布式抽取方法，最后演变为“喷嘴 (spout)”和“bolts(螺拴)”的思想。这两个概念的特点是内在并行处理的，类似于Hadoop。Bolts对于需要处理的流进行订阅，然后指 出之后的流

该如何进行分区处理。进行一番实践后，我决定在这基础上进行进一步改进，于是我在推特上发布了新的构建思路--Storm，结果反应非常热烈。

随后，对中间消息的依赖让我觉得，如果有个更好的办法，消除对中间消息传递的依赖，会更能提高工作效率。于是我尝试使用一种基于随机数和异或逻辑的算法，它仅需要20个字节的长度来跟踪每个spout数组。至此，Storm的雏形基本成型，我隐约感觉这将是一项伟大的工程。

第一个版本

接下来的5 个月时间里，Storm的第一个版本诞生了。从一开始，开源是我觉得应该去做的事情。我用Java进行Storm API的编写，而部署端使用Clojure。事实证明，这会带来更高的软件生产力和工作效率。此外，考虑到兼容性和便利性，我希望Storm平台是能与主 流编程语言无缝工作的，这样人们使用Storm时，可以使用自己熟悉的语言，而不必重写代码。

作为Hadop的 重度使用者，Hadoop的僵尸worker模式会浪费不少资源。于是，在Storm设计中，我把累赘的使用完毕的worker在其首次出现的地方就直接 销毁。另外，一旦Hadoop的JobTracker出现问题，正在运行的作业将会被终止。如果有一项作业已经运行了多天却因此而被迫终止，这无疑是非常 令人沮丧的。所以在Storm中我引入了“process fault-tolerant(进程容错)”机制，简单说，就是即使Storm守护进程被销毁，重启该进程不会对正在运行的拓扑产生影响。

在Jason Jackson的帮助下，在AWS上我们创建了Storm自动部署机，这对Storm的开发过程带来极好的帮助，同时使得我可以在不同大小和配置的机器集群中进行测试。

开源Storm

2011年7月，与Twitter就收购事宜达成一致，我们正式加入Twitter这个大品牌。同时，我马上着手推出Storm。

要想成功发布开源软件，方法有两个。一是把事情“搞大”，尽可能地增加作品的曝光度与关注度，不过缺点是一旦与人们预期不符合，进一步合作意愿可能从此消失。二是静悄悄地发布，慢慢地培养用户群，不足之处是如果人们发现不过如此，很可能从此不再关注。对这两点进行分析论证后，我决定透过开交流会的方式进行宣传和发布。事后证明，这是正确的做法。在正

式发布当天，Github上的浏览和关注数很快就超过了1000次，同时成为Hacker New上排名首位的项目。

Storm的技术演变

初创企业的需求与大型成熟企业的需求截然不同，加入Twitter 后，我们对此有越来越深刻的认识。在Twitter中，人们希望能够直接使用Storm，而把其他的运营工作交给别人。这意味着，我们需要打造一个大型的 共享集群，能够独立运行大量的程序，并确保程序间不会成为彼此的资源争夺者，这一般被称为“多用户架构”。

由于共享的性质，我们发现人们喜欢获取最大程度的资源来运行程序，而实际上很多时候是供大于求。于是，我开发了“isolation scheduler(分离调度表)”。这个机制一来鼓励人们更有效地使用资源，二来允许单个集群共享负载量。

随着Twitter平台的发力，越来越多的开发者成为Storm的用户。出于对系统性能监控需求的考虑，我们提供了相应的指标API，方便人们把相关指标数据与自己喜好的监控系统进行整合。

另外一个较大的技术进步是开发了Trident(三叉戟)，一个基于Storm的微批处理API，提供了精确的一次性语义处理能力，提高了对新用例的处理能力。

离开Twitter，投入Apache怀抱

在2013 年，我离开Twitter发展自己的创业公司，但对于Storm，我始终是开发环节和发展的中心。接下来的数月里，在再次认识以个人为中心模式的种种不足 后，我想是时候采用共识驱动的开发模式了。我希望Storm能成为一项长久的有活力的工程。争取Apache基金会的帮助，投入Apache的怀抱，是非 常合适的选择。Apache有足够的品牌影响力，是个强大的合法的基金会，最关键的是能最大程度帮助Storm过渡到共识驱动模式。随着我作为发展瓶颈的 问题完美解决后，Storm驶入发展的快车道。Storm开发社区活跃度与日俱增，越多越多的新想法新思路被发掘被采纳，在其哇哇坠地的三年后，在 2014年9月成为官方的顶级项目，这就是群众的力量。

写在最后

Storm的成功让我深刻认识到，打造一个成功的项目不仅仅要能帮助人们解决实际问题，还必需兼顾文档，推广，支持社区等方方面面。特 别是项目初期，我们需要有自己的创造性思维，来确保项目被成功推广。此外，自己也需要充分利用时间，来为人们解答疑难，编写支持文档，形成良好的交流氛 围。最后，我们有时候需要做个清醒的旁观者，看有什么会制约项目发展，在适当的时候做出果断的选择，该抽离就抽离，群众的力量总比一个人战斗来得高效长 效。

译文链接：http://www.csdn.net/article/2014-10-13/2822072-Apache-Storm

原文链接：http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html

王帅：深入PHP内核（三）——内核利器哈希表与哈希碰撞攻击

作者：王帅

【导读】王帅在海量分布式Web系统有超过8年沉淀，主导过多个大型系统的架构设计，目前在腾讯企业SaaS团队。

PHP内核系列文 章，是作者在PHP领域实践中，把相关原理性的知识，通过更便于理解的方式，系统整理出来分享给读者。希望通过PHP原理性的轻量解读，对这门Web领域 最热门技术的优秀架构分析解构，让更多的人不断的深入了解语言的原理本身，更容易定位、理解一些问题背后原因，更游刃有余的做基础架构设计。同时希望影响 更多的人才投入Web开源领域，不仅是应用和学习一门技术、组件，同样能够贡献更多高质量组件，像战国时期的百家争鸣一样，PHP开源界花开遍地。

作 者一直倡导技术的深入学习就像职业篮球训练，80%的时间都是基本功的训练，球场上实际战术的练习只是基本功的应用。同样的，学习PHP语言本身的特性， 应当是每个PHP领域工程师所掌握、理解的，至于系统的架构设计也是基于对Linux、Mysql、Nginx等原理机制足够理解后，战术性的使用。

深入PHP内核（三）——内核利器哈希表与哈希碰撞攻击

在 PHP的Zend Engine（下面简称ZE）中，有一个非常重要的数据结构——哈希表（HashTable）。哈希表在ZE中有非常广泛的应 用，PHP的复杂数据结构中数组和类的存储和访问就是用哈希表来组织，PHP语言结构中的常量、变量、函数等符号表也是用它来组织。

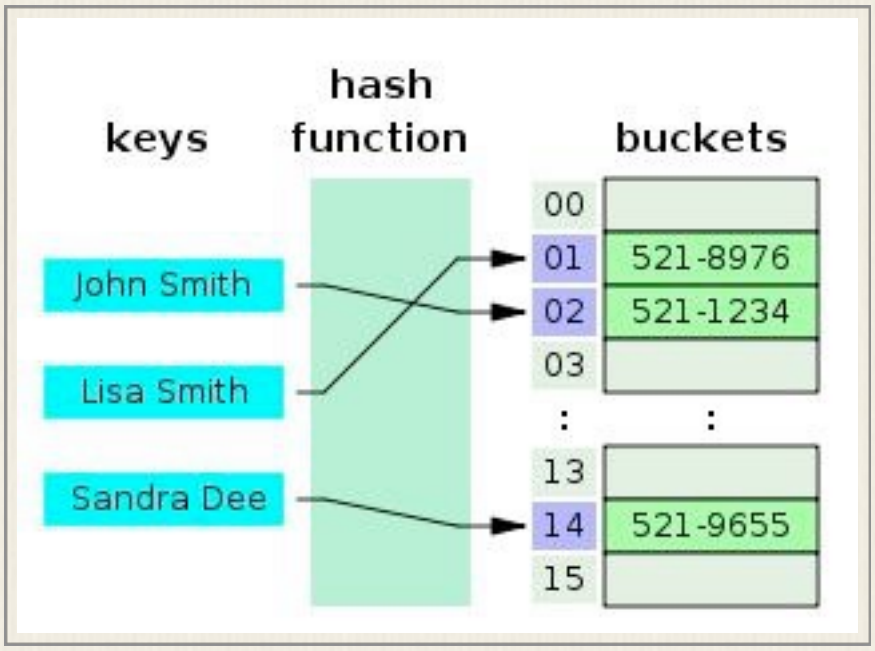
1. 哈希表的基本概念

什么是哈希表呢？哈希表在数据结构中也叫散列表。是根据键名经过hash函数计算后，映射到表中的一个位置，来直接访问记录，加快了访问速度。在理 想情况下，哈希表的操作时间复杂度为O(1)。数据项可以在一

个与哈希表长度无关的时间内，计算出一个值hash(key)，在固定时间内定位到一个桶 (buck-
et，表示哈希表的一个位置)，主要时间消耗在于哈希函数计算和桶的定位。

在分析PHP中HashTable实现原理之前，先介绍一下相关的基本概念：

如下图例子，希望通过人名检索一个数据，键名通过哈希函数，得到指向bucket的指针，最后访问真实的bucket。



键名(Key)：在哈希函数转换前，数据的标识。

桶(Bucket)：在哈希表中，真正保存数据的容器。

哈希函数(Hash Function)：将Key通过哈希函数，得到一个指向bucket的指针。MD5，SHA-1是我们在业务中常用的哈希函数。

哈希冲突(Hash Collision)：两个不同的Key，经过哈希函数，得到同一个bucket的指针。

2. PHP的哈希表实现原理

哈希表的结构：

1. *Zend/zend_hash.h*

```
2.  typedef struct _hashtable {
3.      uint nTableSize;                //哈希表的长度，不是元素个数
4.      uint nTableMask;                //哈希表的掩码，设置为nTableSize-1
5.      uint nNumOfElements;            //哈希表实际元素个数
6.      ulong nNextFreeElement;         //指向下一个空元素位置
7.      Bucket *pInternalPointer;       //用于遍历哈希表的内部指针
8.      Bucket *pListHead;              //哈希表队列的头部
9.      Bucket *pListTail;              //哈希表队列的尾部
10.     Bucket **arBuckets;             //哈希表存储的元素数组
11.     dtor_func_t pDestructor;        //哈希表的元素析构函数指针
12.     zend_bool persistent;           //是否是持久保存，用于pmalloc的参数，可以持久存储在内存中
13.     unsigned char nApplyCount;      // zend_hash_apply的次数，用来限制嵌套遍历的层数，限制为3层
14.     zend_bool bApplyProtection;     //是否开启嵌套遍历保护
15. #if ZEND_DEBUG
16.     int inconsistent;
17. #endif
18. } HashTable;
```

1）nTableSize 哈希表的大小。最小容量是2^3(8)，最大容量是2^31(2147483648)。当如果进行一次操作后发现元素个数大于nTableSize，会申请当前nTableSize * 2的空间。假设当前nTableSize为8，当插入元素达到9个的时候，会申请 nTableSize=16的空间。

2）nTableMask 为nTableSize-1，用于调整最大索引值。当哈希后值大于索引值时候，把这个值映射到索引值范围内。

3）nNumOfElements HashTable中的个数。数组操作中，sizeof和count函数获取的是这个值。

4）nNextFreeElement 下一个空元素的地址。

5）pInternalPointer 存储了HashTable当前指向的元素的指针，当我们使用一些内部循环函数的时候会用到这个指针比如reset(), current(), prev(), next(), foreach(), end()。相当于游标。

6）pListHead和pListTail则具体指向了该哈希表的第一个和最后一个元素，对应就是数组的起始和结束元素。哈希表的pListHead、 pListTail与Bucket的pListNext、pListLast维护了一个哈希表中Bucket的双向链表，按照插入的先后顺序，用于哈希表 的遍历。

7）arBuckets 实际存储Buckets的数组。

8）pDestructor 是一个析构函数，当某个值被从哈希表删除的时候会触发此函数。他还有一个主要作用是用于变量的GC回收。在PHP里面GC是通过引用计数实现的，当一个变量的引用计数变为0，就会被PHP的GC回收。

9）persistent 定义了hashtable是否能在多次request中获得持久存在。

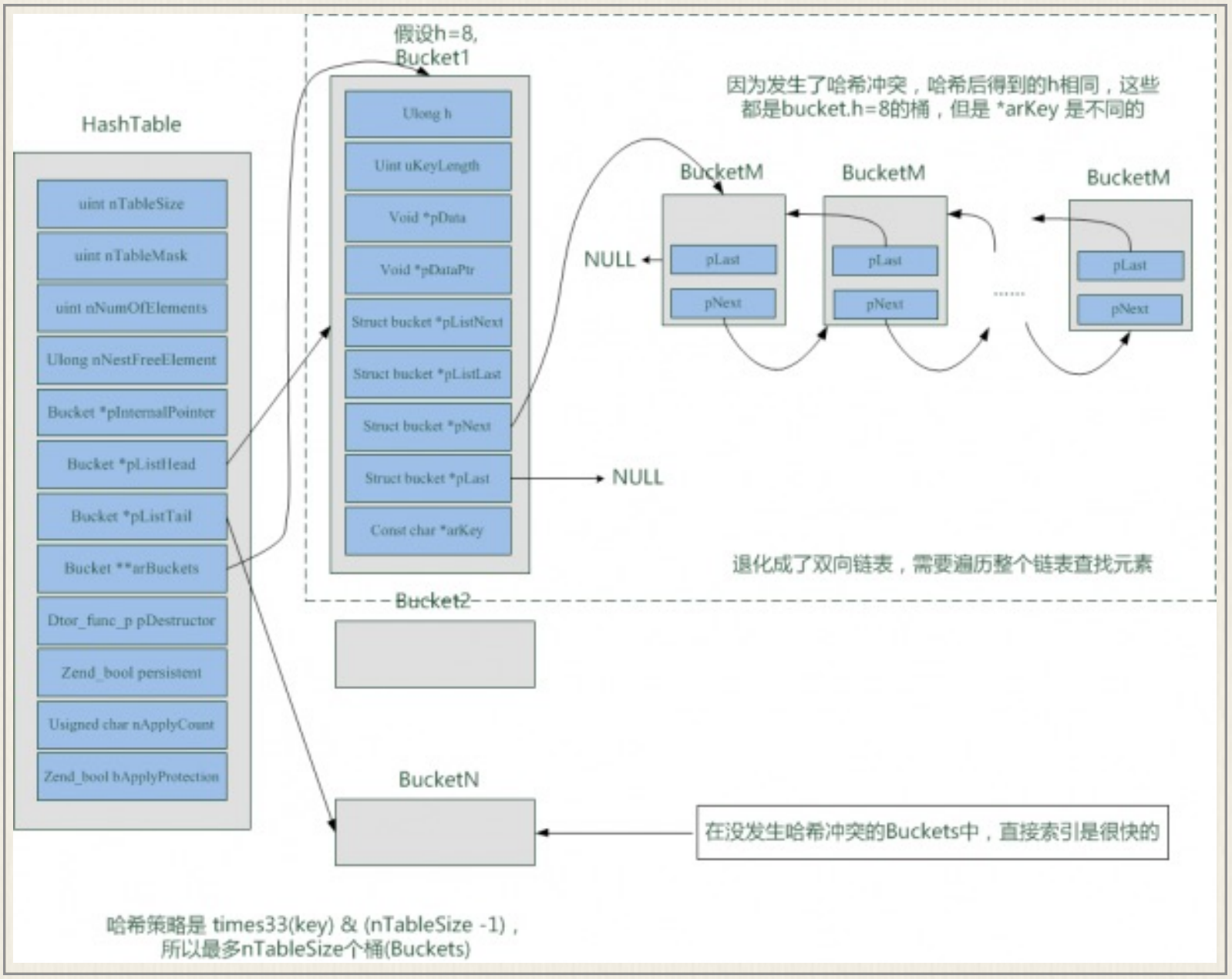
10）nApplyCount 和 bApplyProtection 是用来防止嵌套遍历的。

11）inconsistent 是在调试模式下捕获对HT不正确的使用。

Bucket的结构：

```
1.  typedef struct bucket {
2.      ulong h;                        //数组索引的哈希值
3.      uint nKeyLength;                //索引数组为0，关联数组为key的长度
4.      void *pData;                    //元素内容的指针
5.      void *pDataPtr;                 // 如果是指针大小的数据，用pDataPtr直接存储，pData指向pDataPtr
6.      struct bucket *pListNext;       //哈希链表中下一个元素
7.      struct bucket *pListLast;       //哈希链表中上一个元素
8.      struct bucket *pNext;           //解决哈希冲突，变为双向链表，双向链表的下一个元素
9.      struct bucket *pLast;           //解决哈希冲突，变为双向链表，双向链表的上一个元素
10.     const char *arKey;              //最后一个元素key的名称
11. } Bucket;
```

通过下图来表示HashTable的原理：



我们先来看一下，ZE是如何创建一个hash表的。创建并初始化一个Hash比较容易，调用_zend_hash_init函数。PHP的哈希表最 小容量8(2^3)，最大容量是0x80000000(2^31，即2147483648)。nTableSize会按照2的整数次幂圆整来增加，直到超 过预设值的nSize。

Zend/zend_hash.c

1. *ZEND_API int _zend_hash_init(HashTable *ht, uint nSize, hash_func_t pHashFunction, dtor_func_t pDestructor, zend_bool persistent ZEND_FILE_LINE_DC)*

```
2.    {
3.        uint i = 3;
4.
5.        SET_INCONSISTENT(HT_OK);
6.
7.        if (nSize >= 0x80000000) {
8.            /* prevent overflow */
9.            ht->nTableSize = 0x80000000;
10.       } else {
11.           while ((1U << i) < nSize) {
12.               i++;
13.           }
14.           ht->nTableSize = 1 << i;
15.       }
16.
17.       /* 省略哈希表初始化步骤 */
18.
19.       return SUCCESS;
20.   }
```

1) *ht 是哈希表的指针，这里既可以传入一个已存在的HashTable， 也可以通过内核宏ALLOC_HASHTABLE(ht)来自动申请一块 HashTable内存。ALLOC_HASHTABLE(ht)相当于ht=emalloc(sizeof(HashTable))

2) nSize 哈希表能拥有的最大数量。通过预先申请好内存的方式，减少哈希表rehash操作。

3) pHashFunction 自定义哈希函数的钩子

4) pDesctruc-
tor 哈希表析构的回调函数，当删除一个哈希表的时候，会调用。

5) persistent 对应HashTable.persistent，当设置为true的时候，不会在RSHUTDOWN阶段自动销毁。

我们通过更新哈希表的操作方式，来分析哈希表的操作机制：

[php] view plaincopy

```
1.  h = zend_inline_hash_func(arKey, nKeyLength);
2.  nIndex = h & ht->nTableMask;
3.
4.  p = ht->arBuckets[nIndex];
5.  while (p != NULL) {
6.      if (p->arKey == arKey ||
7.
  ((p->h == h) && (p->nKeyLength == nKeyLength) && !memcmp(p->arKe
y, arKey,       nKeyLength))) {
8.      if (flag & HASH_ADD) {
9.          return FAILURE;
10.     }
11.
```

```
12.     /* 省略 */
13.
14.     UPDATE_DATA(ht, p, pData, nDataSize);   // 找到h 和 Key都相等的Buckets，说明需要更新
15.     /* 省略 */
16.     }
17.     p = p->pNext;   // 这里说明有哈希冲突，按照Buckets[nIndex]的链表找下去
18. }
19.
20. /* 省略 */
21. p->nKeyLength = nKeyLength;
22. INIT_DATA(ht, p, pData, nDataSize);    // 把Bucket.pData数据更新
23. p->h = h;
24. CONNECT_TO_BUCKET_DLLIST(p, ht->arBuckets[nIndex]);    // 挂到
25. if (pDest) {
26.     *pDest = p->pData;
27. }
28.
29. HANDLE_BLOCK_INTERRUPTIONS();
30. CONNECT_TO_GLOBAL_DLLIST(p, ht);
31. ht->arBuckets[nIndex] = p;
32. HANDLE_UNBLOCK_INTERRUPTIONS();
```

33.

34. ht->nNumOfElements++;

35. ZEND_HASH_IF_FULL_DO_RESIZE(ht); /
* 如果哈希表满了，重新散列，这里有一定开销 */

1) 通过哈希算法 times33(Key) & (nTableSize-1) ，生成Key对应的哈希值A，获取arBuckets[A]的值

2) 判断arBuckets[A]是否存在，如果存在而且没有哈希冲突，进行数据update(UPDATE_DATA)。如果存在但是Key不相同说明有哈 希冲突，在arBuckets[A]链表中寻找Key是否存在，如果存在，执行update操作(UPDATE_DATA)

3) 如果arBuckets[A]不存在，创建新的arBucket[A](INIT_DATA)。或哈希冲突情况下，在arBuckets[A]的链表中找 不到Key。创建新的bucket(INIT_DATA)，并把新的buckets放在arBucket[A]链表头

4) 维护哈希表的逻辑链表(CONNECT_TO_GLOBAL_DLLIST)。

5) 如果发现新插入元素已经超过HashTable的nTableSize，自动扩容至2倍nTableSize，重新哈希后维护新的HashTable。

3. PHP使用的哈希函数

PHP的哈希表是用Times33哈希算法，又称为DJBX33A。这是一个使用比较广泛的对字符串的哈希算法，计算速度快，散列均匀，Perl和Apache都使用了这个算法。算法原理就是不断的乘以33，其算法原型如下:

[php] view plaincopy

1. *hash(i) = hash(i-1) * 33 + str[i]*

为 什么是33呢？对于33这个数，DJB注释中是说，1到256之间的所有奇数，都能达到一个可接受的哈希分布，平均分布大概是86%。而其中 33，

17，31，63，127，129这几个数在面对大量的哈希运算时有一个更大的优势，就是这些数字能将乘法用位运算配合加减法替换，这样运算速度会 更高。gcc编译器开启优化后会自动将乘法转换为位运算。PHP实际算法如下：

[php] view plaincopy

```
1.
 static inline ulong zend_inline_hash_func(const char *arKey, uint nKeyLen
gth)
2.  {
3.      register ulong hash = 5381;
4.
5.      /* variant with the hash unrolled eight times */
6.      for (; nKeyLength >= 8; nKeyLength -= 8) {
7.          hash = ((hash << 5) + hash) + *arKey++;
8.          hash = ((hash << 5) + hash) + *arKey++;
9.          hash = ((hash << 5) + hash) + *arKey++;
10.         hash = ((hash << 5) + hash) + *arKey++;
11.         hash = ((hash << 5) + hash) + *arKey++;
12.         hash = ((hash << 5) + hash) + *arKey++;
13.         hash = ((hash << 5) + hash) + *arKey++;
14.         hash = ((hash << 5) + hash) + *arKey++;
15.     }
```

```
16.     switch (nKeyLength) {
17.
 case 7: hash = ((hash << 5) + hash) + *arKey++; /* fallthrough... */
18.
 case 6: hash = ((hash << 5) + hash) + *arKey++; /* fallthrough... */
19.
 case 5: hash = ((hash << 5) + hash) + *arKey++; /* fallthrough... */
20.
 case 4: hash = ((hash << 5) + hash) + *arKey++; /* fallthrough... */
21.
 case 3: hash = ((hash << 5) + hash) + *arKey++; /* fallthrough... */
22.
 case 2: hash = ((hash << 5) + hash) + *arKey++; /* fallthrough... */
23.         case 1: hash = ((hash << 5) + hash) + *arKey++; break;
24.         case 0: break;
25.         EMPTY_SWITCH_DEFAULT_CASE()
26.     }
27.     return hash;
28. }
```

PHP在哈希算法上有所优化，使用了(hash<<5)+hash，效率有所提高。至于hash的初始值为什么为一个大素数5381，要数学上来解释了，不是很理解。

4. 操作哈希表的内部函数

PHP的变量符号表是通过哈希表来维护，首先介绍一下再PHP扩展中如何创建一个新的变量。PHP变量介绍，请看我上一篇文章，《深入PHP内核 - 弱类型变量原理探究》。

[php] view plaincopy

```
1.  ZEND_FUNCTION(variable_creation)
2.  {
3.      zval *new_var1, *new_var2, *new_var3; //创建两个新的变量容器
4.  char *string_contents = "This is a new string variable";
5.
6.      MAKE_STD_ZVAL(new_var1);  //为new_var1申请空间并初始化
7.      MAKE_STD_ZVAL(new_var2);
8.
9.      ZVAL_LONG(new_var1, 10);  //设置new_var1并赋值为long
10.     ZVAL_LONG(new_var2, 5);
11.
 ZVAL_STRINGL(new_var3, string_contents, sizeof(string_contents), 0);
 //设置new_var3为字符串
12.
13.
 ZEND_SET_SYMBOL(EG(active_symbol_table), "local_variable", new_var1);  //设置long_variable为函数variable_creation的局部变量
```

```
14.
ZEND_SET_SYMBOL(&EG(symbol_table), "global_variable", new_var2
);      //设置global_variable为全局变量
15.
16.     zend_hash_update(
17.         &EG(symbol_table),
18.         "new_var3",
19.         strlen("new_var3") + 1,
20.         &new_var3,
21.         sizeof(zval *),
22.         NULL
23.     );
24.
25.     RETURN_NULL();
26. }
```

这里的zend_hash_update会更新变量符号表。PHP的数组也是用哈希表来维护，下面通过操作一个array来解释如何使用哈希表来才做数组。

增加一个关联数组：

[php] view plaincopy

```
1.    zval *new_array, *new_element;
2.   char *key = "element_key";
3.
4.   MAKE_STD_ZVAL(new_array);
```

```
5.  MAKE_STD_ZVAL(new_element);
6.
7.  array_init(new_array);
8.
9.  ZVAL_LONG(new_element, 10);
10.
11. if(zend_hash_update(new_array->value.ht, key, strlen(key) + 1, (void *)&new_element, sizeof(zval *), NULL) == FAILURE)
12. {
13.     // do error handling here
14. }
```

增加一个索引数组：

[php] view plaincopy

```
1.  zval *new_array, *new_element;
2.  int key = 2;
3.
4.  MAKE_STD_ZVAL(new_array);
5.  MAKE_STD_ZVAL(new_element);
6.
7.  array_init(new_array);
8.
9.  ZVAL_LONG(new_element, 10);
```

10.

11.

*if(zend_hash_index_update(new_array->value.ht, key, (void *)&new_elem ent, sizeof(zval *), NULL) == FAILURE)*

12. {

13. // do error handling here

14. }

哈希表的增删改查

[php] view plaincopy

1.

*int zend_hash_add(HashTable *ht, char *arKey, uint nKeyLen,void * pData, uint nDataSize, void **pDest);*

2.

*int zend_hash_update(HashTable *ht, char *arKey, uint nKeyLen, v oid *pData, uint nDataSize, void **pDest);*

3.

*int zend_hash_index_update(HashTable *ht, ulong h, void * pData, uint nDataSize, void **pDest);//与zend_hash_update*类似，不过哈希值计算是用*h&TableMask*

4.

*int zend_hash_next_index_insert(HashTable *ht, &nb sp;   &nb sp;   void *pData, uint nDataSize, void **pDest);*

5.

*int zend_hash_find(HashTable *ht, char *arKey, uint nKeyLength,void **pData);*

6.

7.

*int zend_hash_index_find(HashTable *ht, ulong h, void **pData); *

8.

*ZEND_API int zend_hash_exists(const HashTable *ht, const char *arKey, uint nKeyLength)*

9.

*ZEND_API ulong zend_get_hash_value(const char *arKey, uint nKeyLength)*

10.

11.

*ZEND_API void zend_hash_merge_ex(HashTable *target, HashTable *source, copy_ctor_func_t pCopyConstructor, uint size, merge_checker_func_t pMergeSource, void *pParam)*

12. 通过*source*的逻辑双向链表，遍历*source*插入*target*

13.

*ZEND_API void zend_hash_copy(HashTable *target, HashTable *source, copy_ctor_func_t pCopyConstructor, void *tmp, uint size)*

哈希表的遍历

[php] view plaincopy

1. *ZEND_API int zend_hash_get_pointer(const HashTable *ht, HashPointer *ptr)*

2. *ZEND_API int zend_hash_set_pointer(HashTable *ht, const HashPointer *ptr)*

3. *ZEND_API void zend_hash_internal_pointer_reset_ex(HashTable *ht, HashPosition *pos)*

4. *ZEND_API void zend_hash_internal_pointer_end_ex(HashTable *ht, HashPosition *pos)*

5. *ZEND_API int zend_hash_move_forward_ex(HashTable *ht, HashPosition *pos)*

6. *ZEND_API int zend_hash_move_backwards_ex(HashTable *ht, HashPosition *pos)*

数组操作函数reset(), each(), current(), next()会用这些函数来实现。

比较，排序

[php] view plaincopy

1. *ZEND_API int zend_hash_sort (HashTable * ht , sort_func_t sort_func , compare_func_t compar , int renumber TSRMLS_DC)*

2.

3.

*ZEND_API int zend_hash_minmax (const HashTable * ht , compare_func_t compar , int flag , void ** pData TSRMLS_DC)*

4.

5.

*ZEND_API int zend_hash_compare (HashTable * ht1 , HashTable * ht2 , compare_func_t compar , zend_bool ordered TSRMLS_DC)*

哈希表有一套排序算法。sort(), asort(), resort(), arsort(), ksort(), krsort()

详细请见： http://php.net/manual/en/array.sorting.php

5. 哈希冲突（Hashtable Collisions）

因为任何一个哈希表的长度都是有限制的，所以一定会发生键名不同，hash函数计算后得到相同的bucket位置。也就是 key1 != key2，但是HASH(key1) = HASH(key2)。如下图2，在发生哈希冲突时（Hash Collision），最坏情 况下，所有的键名全部冲突，哈希表会退化成双向链表，操作时间复杂度为O(n)。

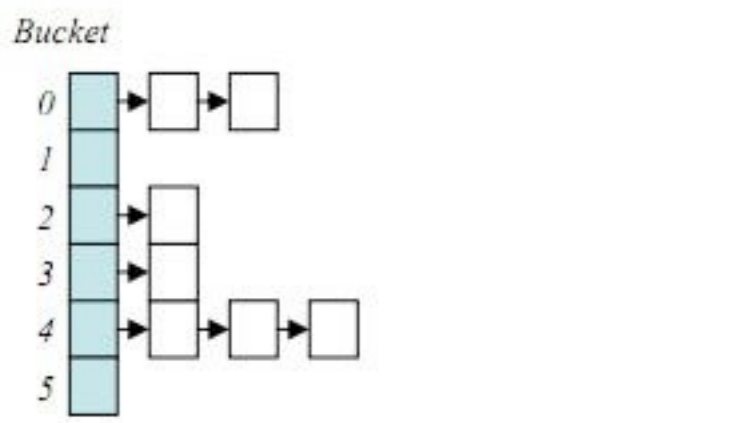


Figure 1: Normal operation of a hash table.

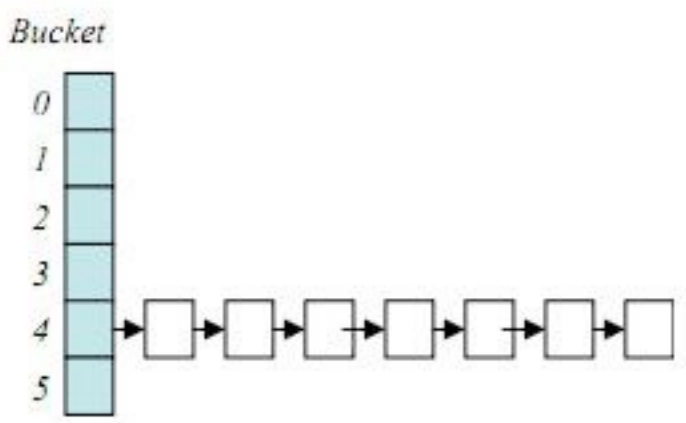


Figure 2: Worst-case hash table collisions.

当发生了哈希冲突，会把当前bucket插入到哈希值所在链表的第一位，并插入HashTable的逻辑链表。

6. 哈希碰撞攻击及解决

在去年发现了PHP的哈希碰撞攻击漏洞，PHP5.3.9以下的版本都会受影响。我们在业务压力很重的情况下，还是最短时间内把运营服务器全部更新到5.3.13以上，防止通过PHP的哈希碰撞进行拒绝服务攻击。

如何哈希碰撞攻击呢？运用哈希冲突。在我们对PHP哈希算法足够了解以后，通过精心构造，可以让PHP的哈希表全部冲突，退化成链表，每插入元素时候，PHP都要遍历一遍链表，消耗大量的CPU，造成拒绝服务攻击。最简单的方法是利用掩码规律制造碰撞，我们知道HashTable的长度nTableSize会被圆整为2的整数次幂，假设我们构造一个长度为2^16的哈希表，nTableSize的二进制表示 为：1 0000 0000 0000 0000，而nTableMask = nTableSize – 1 为：0 1111 1111 1111 1111。这样我们只要保证后16位均为0，则与掩码与运算后得到的哈希值全部碰撞在位置0。

[php] view plaincopy

1. *0000 0000 0000 0000 0000 & 0 1111 1111 1111 1111 = 0 *
2. *0001 0000 0000 0000 0000 & 0 1111 1111 1111 1111 = 0*
3. *0010 0000 0000 0000 0000 & 0 1111 1111 1111 1111 = 0*
4. 。。。

以下这个例子就是这个原理的实现，插入65535个数据需要消耗30秒，而正常情况下仅需要0.01秒。

[php] view plaincopy

```
1.  <? php
2.  echo '
3.  ';
4.
5.
$size = pow(2, 16); // 16 is just an example, could also be 15 or 17
6.
7.  $startTime = microtime(true);
8.
9.  $array = array();
10.
for ($key = 0, $maxKey = ($size - 1) * $size; $key <= $maxKey; $key += $
size) {
11.      $array[$key] = 0;
12.  }
13.
14.  $endTime = microtime(true);
15.
16.
echo 'Inserting ', $size, ' evil elements took ', $endTime - $startTime, ' sec
onds', "\n";
17.
18.  $startTime = microtime(true);
19.
20.  $array = array();
```

```
21.  for ($key = 0, $maxKey = $size - 1; $key <= $maxKey; ++$key) {
22.      $array[$key] = 0;
23.  }
24.
25.  $endTime = microtime(true);
26.
27.
echo 'Inserting ', $size, ' good elements took ', $endTime - $startTime, ' se
conds', "\n";
28.  ?>
```

结果是

[php] view plaincopy

```
1.    Inserting 65536 evil elements took 32.726480007172 seconds
2.
Inserting 65536 good elements took 0.014460802078247 seconds
```

文章来源：http://nikic.github.io/2011/12/28/Supercolliding-a-PHP-array.html

对于哈希碰撞攻击有2中常见形式：通过POST攻击或通过反序列化攻击。PHP会自动把HTTP包中POST的数据解析成数组$_POST，如果我们构造一个无限大的哈希冲突的值，可以造成拒绝服务攻击。

PHP5.3.9+是通过增加一个限制来尽量避免被此类攻击影响：

[php] view plaincopy

1. - *max_input_vars* - 指定 *GET/POST/COOKIE* 的最大输入变量数。默认是*1000*。

反序列化同样是利用数组的哈希冲突，如果POST的数据有字段为数组serialize后的值，或数组json_encode后的值，在 unserialize或json_decode后，会有可能造成哈希碰撞攻击。解决方法，尽量避免在公网上以数组的序列化形式传递数据，如果不可避免， 请使用私有协议(TLV)增加供给难度，或使用加密协议(HTTPS)防止中间人攻击。

7. 总结

PHP的哈希表采用times33的哈希算法，通过HashTable数据结构维护Buckets，当有哈希冲突的时候，会将元素插入到该 Buckets前形成双向链表。同时为了方便遍历，HashTable也会维护逻辑双向链表（按照插入顺序），通过内部游标指针可以遍历 Hashtable。PHP的变量符号表、常量符号表和函数都是用哈希表维护，PHP的数组类型变量也是通过哈希表维护。

哈希表容易遭到哈希碰撞攻击，请更新PHP版本到5.3.9以上，可以解决POST数据的攻击问题；反序列化（把序列化字符串还原为Array）的哈希碰撞攻击，到目前位置PHP官方还没有彻底解决这个问题，请尽量避免用户篡改数据和中间人攻击。

原文链接：http://www.csdn.net/article/2014-10-16/2822134

专访徐宜生：坚决不做代码搬运工！

作者：单明珠

徐宜生（博客）， 毕业于上海海事大学网络工程专业，为了学习到更多技术知识，从对日外包公司跳槽至TCL工作，目前主要负责国外一些移动运营商的手机系统订制工作。在进入 TCL工作之前，工作内容与Android没有太大的关系，所以基本上都是在自学Android知识。到TCL后，能够接触到开发系统的系统层，他表示在 公司学到的要比自学学的更好更快。

对技术感兴趣的同学，最好不要进日企

CSDN：请和大家介绍下你和目前所从事的工作，以及你的学习经历。

徐宜生：大家好，我是徐宜生，是一个热爱编程、喜欢分享的极客，毕业于上海海事大学网络工程专业。在大学期间就喜欢上了编程带给自己的成就感，热衷于钻研技术，从最早接触的Java EE到后来的Android，一直都是兴趣在驱使着我学习。我的第一份工作与Android没有太大的关系，所以基本上都是在自学，从最基础的学起，到后面慢慢的做一些项目、实战，一步步走来。现在在TCL，负责对运营商的Android系统进行订制。

CSDN：第一份工作是在对日外包公司，谈下在那的工作情况吧?

徐宜生：由于找工作的时候没有考虑清楚，稀里糊涂就进了一家对日外包公司，对于刚毕业的同学来说，对日外包也还算是一份不错的工作， 因为工作不会有太大的技术难度，可以有个缓冲过渡期，让你慢慢适应走向社会。日本公司有一个很大的特点就是流程特别规范，在那可以接触到完整的开发流程， 每一个阶段、每一个项目，都会有很详细的文档资料，而且公

司对文档资料的要求也特别严。所以，对那些想要学日语或者对技术不是很感兴趣的同学来说，进日企 也是一个不错的选择。

但是，那些对技术感兴趣的同学，最好不要进这样的公司。因为对日项目一般都会用日本人自己搭建的框架，你要做的就是用这个框架去照着日本人给的例子 去写，甚至有些项目的详细设计也是日本人做好的，你要做的就只是把他详细设计里面的伪代码翻译成代码，而它核心的逻辑、框架，你根本接触不到。这样的话， 你就是个真正的码农，一个代码搬运工。而且，日企的技术一般都很落后，基本上不会用很新的技术，几乎都是在维护之前做过的东西。所以，对那些想要提高自身 技术的程序员们，就不要考虑来这样的公司了。也正是因为这样，我选择了离职。

CSDN：为什么会跳槽至TCL，现在主要负责什么工作?

徐宜生：在对日外包工作了一年多时间后，我越来越觉得，如果继续在这里待下去，不仅技术得不到积累，也是在荒废时间。因此在这里也建议学弟学妹们以我为前车之鉴，在选工作的时候，一定要先认清自己的兴趣方向。程序员这样一个行业，如果没有兴趣，那真的很难有进步空间。

离职到TCL后，主要负责国外一些移动运营商的手机系统订制工作，目前在Framework Team，负责维护、修改工厂用的测试软件、解决Framework层的一些技术问题等。

学东西要知其然，也要知其所以然

CSDN：听说你大学的时候专业是网络工程，而现在从事编程工作，你是如何处理这种转变的呢?

徐宜生：其实大学对专业的区分度也不是太大，毕竟都是计算机专业，学的东西也有很多相同的地方，所以这里也想对那些大学的新生讲，不 要因为进了自己不喜欢的专业就不努力学习，大学学习的重点是教会学习的方法和方向。像我，大学学的网络专业，让我对网络有了比较清晰的认识，这对网络编程 来说也是很有帮助的。

CSDN：据我的了解，在开发系统方面，你自学的都是在应用层，到TCL后接触到更多的都是系统层，其中有哪些心得值得与我们分享?

徐宜生：嗯，要学一门技术，首先要学会使用它，因此，在自学的时候，肯定要从它的应用层开始，然后再慢慢的去深入了解，知其然，然后知其所以然。到TCL后，公司把我分到Framework Team，后来感觉这样挺好，因为在应用层我自学了那么长时间，基本上也都了解（不敢说精通），最起码知道方向。这样写应用，就好比搭积 木，Android提供了各种各样的积木，但你只是单纯的会用，却不知道它是如何实现的。好奇心是人的天性，相信大家在用Android的控件去实现某个 效果的时候，一定也会去想，它是怎么实现的呢？从另一个角度来说，就像高中背物理公式一样，老师不会给你一个公式让你去背去用，而是先给你推导一遍，这样 就算你忘了公式，自己也能很快推导出来。其实编程也是一样，因为你不可能把所有的东西都记住，就像今天在流行Android 4.4，明天也许就流行L了，虽然它一直在变，但思想不会有太大变化，就像许多大牛说的，编程语言的原理都是一样的，只要精通一门，再学习其他语言就很轻 松了。

所以说，不管是上层开发者还是底层开发者，如果能了解一些原理技巧，对工作、自身知识的掌握，都是很有帮助的。其实，我们并不要求每一个上层开发者都能像老罗那样对底层如数家珍，只要知道大致的流程分析、实现原理后，你的技术能力肯定会更上一层楼。

CSDN：在TCL工作，能学到更多的开源框架技术么？会不会遇到什么问题?

徐宜生：大公司在项目上一般不会用太多现成的框架，但在项目管理和开发上，会使用框架，比如代码搜索引擎用OpenGrok；版本管 理工具Git用Gerrit；内部论坛用WordPress。在项目方面，由于我们是直接与国外运营商接轨，所以应用层的开发不是太多，也就用不到现在市 面上很多开源项目。

刚接触手机系统开发的时候，确实会遇到很多问题，特别是像我这样之前以应用层为主的开发者，就像我前面说的那样，多看看底层，深度理解设计思想和实 现过程是很有必要的。刚开始看的时候或许会比较吃力，需要多请教同事，多和同事讨论，才能找到解决的方法。有时，不妨也和硬件的

同事们打下交道，因为术业 有专攻，从他们身上总能学到很多知识。此外，自己也要多查资料。

开源的本质是交流，我们要多分享

CSDN：听说你模仿了两款Android 游戏（2048和拼图），并开发一款仿iOS Assistanttouch的工具，现在在做公司的内部培训辅助系统。在研发过程中又遇到什么问题，又有哪些收获?

徐宜生：有想法的时候，一定要去做，不能想的太多，动手做才是最重要的，当然，构思项目架构、逻辑关系，这些还是需要要的。但只有 在实践中，才能最大程度的调动知识储备，才能知道自己在哪方面掌握的还不够。以前自己做Android项目的时候，遇到的最大问题就是UI设计的问题，对 于没有美工的程序员来说，这个一直都是泪点。

技术上的难点，基本上通过各类文档、API DOC都能找到相应的解决办法，没有什么技术上的创新，只是一个对知识的沉淀、积累的过程。写这些项目的目的是在于把所学到的知识运用到实际中去，完成对 知识的积累，加深自己的理解。而像现在在给公司内部做的这样一个培训系统来说，就不能像以前自己做项目那样完全按照自己的想法来做，你要听取其他人的意见和建议，而且整个项目的流程、逻辑也需要更多的推敲、完善，相比前面几个项目，业务上的逻辑更多、更复杂了。

CSDN：能否谈下你对开源的理解，以及对国内开源技术和产品的看法?

徐宜生：人就是站在前人的肩膀上不断进步的，开源才是技术进步的第一动力，与他人分享技术、经验，不仅可以让你学到别人的经验，更能 让你的项目更加的完善，毕竟一个人的能力、思维是有限的，大家一起帮你改善，反过来你的项目又帮助了更多的人，就是这样一种方式，才促使技术不断的进步。

但国内外对开源的理解可能不太一样，毕竟在中国，版权问题都还没能好好解决，而且很多公司也都处于保守的观念中，这跟国内外的文化差异也是很有关系 的，他们认为自己创造的东西凭什么白给别人，这样做当然没有错，但你看特斯拉，它也开源了他们的专利，就目前而言，好像自己的研究成果都白给了别人，但长 期来看，最终受益的肯定还是他们，它引导了一个行业，并在开源的基础上吸纳了更多，这样进步的速度肯定大于它闭门造车的进步速度。

曾有大牛说过，代码无用，思想无价。我们不能一辈子做伸手党，也要学会回馈，开源的本质就是交流，你学了新的东西，当你有新的发现的时候，也要交给更多的人，这才是开源的精神所在。

初学者不要学习太多开源框架，要打好基础

CSDN：对初学开源框架的程序员，你有哪些建议?

徐宜生：我的建议是先不要学习太多太复杂的开源框架，把基础的东西打好，如果你对Android本身都不是很熟悉，那么研究这些框架 你会很吃力，而且也会让你越学越摸不着头脑，框架这个东西，是为了简化开发过程、降低程序耦合度才创造出来的，所以你必须有一定要有足够的代码量，只有写 的多了，才知道为什么要偷懒，才知道人家这样设计的好处是什么，特别是当你也想到了这样的思路时候，那么在它的参考下，不仅是给你指明了方向，更给你铺平 了道路。

另外，建议大家在阅读开源框架的源码的时候，可以由总体到局部，再由局部到整体的方式来看，先了解整个框架运行的流程、架构，再去看相应的模块是如 何实现的，最后再把这些结合起来，看作者是如何设计这些架构的，这样才能学到他的精髓，如果只是一味的去用而不去思考，那么你就很难有所创新了，最后还是 那句话，不管学什么框架，都是这一句话--RTFSC（Read the f**king souce code）。

CSDN：为什么会钻研Andbase开源框架和UI控件呢?

徐宜生：其实每个程序员都会有很多未完成的项目，有时候有了一些idea，就想去做，但是每次都要从头做起，在你实现自己的idea 之前，需要做很多实现idea以外的工作，比如搭建UI的界面，就算是最简单的滑动Tab之类的，也要写好几个类，然后还有数据库的操作辅助类，常用的工 具辅助类，然后才开始写自己的核心逻辑，其实前面这很多东西每个项目都是要经过的，当你做这些事情做多了之后，也就想偷偷懒了，所以那个时候我也在思考， 如何来做一个框架来简化这些工作，后来就在网上找到了一些实现的方法，其中，Andbase这个框架跟我的思路最为相近，所以自然而然就开始学习他的思路 和方法。当然，这样的框架还有很多，Afinal也是一个不错的例子，这些框架需要你对整个Android架构有一定的了解的基础上才能去好好研究，当 然，当你研究多了，对整个Android的了解和自己的能力都会起到很大的帮助。

在应用层，其实UI是非常重要的一件事，在你实战过几个项目后，逻辑处理这方面，就已经基本都能实现了，这时候你再看你做的app，你就会发现，最 大的一个问题就是丑，而且交互差，所以，很自然的你就想去改善你的UI了，而且，UI框架也是Android中很重要的一环。因此，研究UI的开源组件， 你不仅可以改善你自己的界面设计，而且还能让你看见别人的创意，更能让你学到很多Android UI相关的知识，这些东西都是最上层的交互，更能体现技术是如何用于实践、如何表达创意的。

CSDN：你业余时间会做什么？研究新技术?

徐宜生：业余时间会写一些博客，把最近学到的、总结的新知识沉淀下来，在与他人分享的同时，也学习别人的分享。周末会给自己放个假，好好的休息下。不过有时候有想做的App，也会加班加点、通宵达旦的去做，毕竟兴趣是最好的老师，在兴趣的驱动下，才有动力。

原文链接：http://www.csdn.net/article/2014-10-13/2822071